(1)

MASTER'S PROJECT REPORT

DESIGN AND IMPLEMENTATION OF A CONTROLLER AND A HOST SIMULATOR FOR A RELATIONAL REPLICATED DATABASE SYSTEM

SUBMITTED TO:

GRADUATE COMMITTEE
DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF CENTRAL FLORIDA

BY

LEONARD A. LYON

ADVISORY COMMITTEE

DR. ALI OROOJI, CHAIRMAN
DR. M. BASSIOUNI
DR. S. LANG

SUMMER 1987

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ADA198951

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/CI/NR 88- 61 · 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DESIGN AND IMPLEMENTATION OF A CONTROLLER AND A HOST SIMULATOR FOR A RELATIONAL REPLICATED DATABASE SYSTEM	5. TYPE OF REPORT & PERIOD COVERED MS THESIS
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) LEONARD A. LYON	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: UNIVERSITY OF CENTRAL FLORIDA	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS	12. REPORT DATE 1988
	13. NUMBER OF PAGES 167
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) AFIT/NR Wright-Patterson AFB OH 45433-6583	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

DISTRIBUTED UNLIMITED: APPROVED FOR PUBLIC RELEASE

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

SAME AS REPORT

18. SUPPLEMENTARY NOTES

Approved for Public Release: IAW AFR 190-1
LYNN E. WOLAVER [illegible] 19 Jul 88
Dean for Research and Professional Development
Air Force Institute of Technology
Wright-Patterson AFB OH 45433-6583

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

ATTACHED

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

The host simulator program is used to simulate requests generated by a host computer and to verify the proper operation of the controller. The functions performed by the host simulator program will provide the capability to:

(1) Build and display relation templates used to construct requests.

(2) Build a series of request files composed of various RRDS requests.

(3) Display the contents of a specific request file.

(4) Allow a user to edit a request file to add, delete, or modify requests.

(5) Allow selection of a specific request file for processing by the controller program.

(6) Pass requests to the controller individually from a chosen file or generated interactively.

(7) Accept coalesced results from the controller and log them.

1.3. Thesis Overview

We describe the implementation of the host simulator program and the controller program in the remainder of this report. Chapter 2 provides the reader with a detailed description of the implementation of the host simulator program. Chapter 3 describes the implementation of the RRDS controller program. Chapter 4 presents conclusions reached during this project and suggested improvements for future work.

The appendices provide the design documents and source code for the controller and host simulator programs. The BNF syntax of the RRDS data manipulation language, as proposed in the RRDS design, is contained in Appendix A. Appendix B describes the program design language and Jackson Charts used in designing the two programs. The Jackson charts and program design description of the host simulator program and RRDS controller program are presented in Appendices C and D, respectively.

Appendix E contains a user's manual to run the two programs. The specification code for the LEX lexical analyzer and the YACC parser is given in Appendices F and G, respectively. Finally, the C language source code for the host simulator program, the RRDS controller program, and the common routines is provided in Appendices H, I, and J, respectively.

MASTER'S PROJECT REPORT

DESIGN AND IMPLEMENTATION OF A CONTROLLER AND A HOST SIMULATOR FOR A RELATIONAL REPLICATED DATABASE SYSTEM

SUBMITTED TO:

GRADUATE COMMITTEE
DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF CENTRAL FLORIDA

QUALITY INSPECTED 2

BY

LEONARD A. LYON

ADVISORY COMMITTEE

DR. ALI OROOJI, CHAIRMAN
DR. M. BASSIOUNI
DR. S. LANG

SUMMER 1987

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Background

Database machines and parallel database systems are becoming a viable alternative for handling transactions on voluminous databases especially with the declining cost of hardware. Database research has spawned a variety of parallel processing architectures to improve the performance of database machines, ~~[Hans87]~~. However, many of these architectures are based upon unique hardware configurations and technologies including processor-per-track, processor-per-head, and off-the-disk designs, [Hara83]. Alternative backend database system designs have evolved that exploit the power of parallel processing utilizing conventional, off-the-shelf hardware. The relational replicated database system (RRDS) is one such design.

RRDS is a backend database system which will be utilized for processing transactions on voluminous databases, [Hans86a, Hans86b, ~~Hans86c]~~. The complete performance analysis and design of RRDS is currently being carried out by a Ph.D. student at the University of Central Florida. RRDS supports the relational data model and consists of multiple processors running replicated copies of the database management system (DBMS) software on a partitioned database. The general hardware organization, ~~as shown~~ in Figure 1, is composed of a controller and a collection of replicated computers (RCs), which can be implemented as microcomputers or minicomputers.

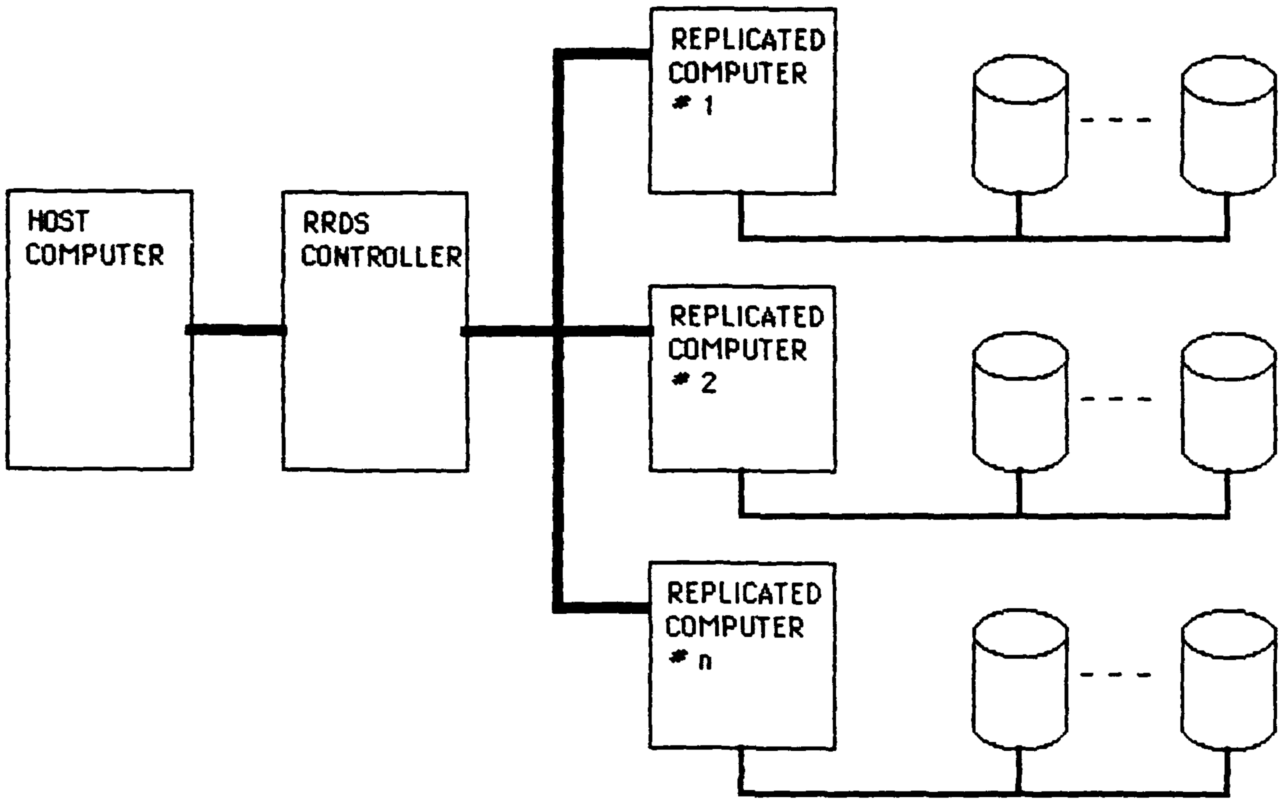


Figure 1: The RRDS Architecture

Each RC maintains a copy of the DBMS software along with separate portions of various database files to enable parallel processing of requests. The controller takes requests from the host computer, parses them, and forwards the valid requests to the RCs via the broadcast bus. Each RC reads the requests from the broadcast bus and processes them on its portion of the database file(s). The results are forwarded to the controller, which coalesces the results and passes them to the host computer.

RRDS supports the complete set of operations in relational algebra, including one-relation operations and two-relation operations. In addition, RRDS provides the following aggregate capabilities: count, sum, max, min, and average. A complete list of RRDS operations is provided in Table 1.a and Table 1.b.

1.2. Project Objectives

The objectives of this project are to design and implement the controller component of RRDS and a host simulator. The controller is the interface between the host computer and the RCs of RRDS. The functions performed by the controller program will provide the capability to:

(1) Accept requests from the host simulator program.

(2) Perform the syntax analysis of the requests.

(3) Forward correct requests to a file (simulated bus).

(4) Simulate RC output by generating relation fragments or aggregate values.

(5) Coalesce the relation fragments with duplicate records removed.

(6) Combine aggregate function values and compute the final result.

(7) Forward the coalesced relation or aggregate value to the host simulator.

One-Relation Operations	Two-Relation Operations
Delete	Difference
Insert	Intersection
Project	Join
Select	Union
Update	
Aggregates	

Table 1.a: The RRDS Operations

Aggregate Operations
Average
Count
Max
Min
Sum

Table 1.b: The RRDS Aggregate Operations

The host simulator program is used to simulate requests generated by a host computer and to verify the proper operation of the controller. The functions performed by the host simulator program will provide the capability to:

(1) Build and display relation templates used to construct requests.

(2) Build a series of request files composed of various RRDS requests.

(3) Display the contents of a specific request file.

(4) Allow a user to edit a request file to add, delete, or modify requests.

(5) Allow selection of a specific request file for processing by the controller program.

(6) Pass requests to the controller individually from a chosen file or generated interactively.

(7) Accept coalesced results from the controller and log them.

1.3. Thesis Overview

We describe the implementation of the host simulator program and the controller program in the remainder of this report. Chapter 2 provides the reader with a detailed description of the implementation of the host simulator program. Chapter 3 describes the implementation of the RRDS controller program. Chapter 4 presents conclusions reached during this project and suggested improvements for future work.

The appendices provide the design documents and source code for the controller and host simulator programs. The BNF syntax of the RRDS data manipulation language, as proposed in the RRDS design, is contained in Appendix A. Appendix B describes the program design language and Jackson Charts used in designing the two programs. The Jackson charts and program design description of the host simulator program and RRDS controller program are presented in Appendices C and D, respectively.

Appendix E contains a user's manual to run the two programs. The specification code for the LEX lexical analyzer and the YACC parser is given in Appendices F and G, respectively. Finally, the C language source code for the host simulator program, the RRDS controller program, and the common routines is provided in Appendices H, I, and J, respectively.

2. IMPLEMENTATION OF THE HOST SIMULATOR

The host simulator program was implemented on a VAX 11/780 computer, operating under the 4.3 Berkeley version of UNIX. The program was written in the C programming language [Kern78] and uses the socket facility of UNIX [UNIX83] to implement the functions described in Chapter 1. The socket facility provides the host simulator program with an interprocess communication channel. The channel is used to send RRDS requests to the controller program and to receive request results from the controller program.

The program is menu driven and prompts the user for all necessary information. There are currently six primary operations that the user can perform via the menu. These operations provide the capability to:

(1) Create a relation template.

(2) Display existing relation templates.

(3) Create a file of RRDS requests.

(4) Display a request file.

(5) Edit a request file.

(6) Run requests from a file or generated interactively from the terminal.

The main menu of the host simulator program is shown in Figure 2, and the operations provided by the program are described in the following sections.

```
**************************************************
******                                      ******
******          HELLO user_name             ******
******                                      ******
******          WELCOME  TO THE             ******
******                                      ******
******      HOST SIMULATOR (LEVEL 1)        ******
******                                      ******
**************************************************

Please select a number for the desired operation.
[1]  CREATE  a relation template.
[2]  DISPLAY relation templates.
[3]  CREATE  a request file.
[4]  DISPLAY a request file.
[5]  EDIT    a request file.
[6]  RUN     a request file.
[0]  QUIT
->
```

Figure 2: The Main Menu of the Host Simulator Program

2.1. Creating Relation Templates

Relation templates are used for the construction of RRDS requests. The templates provide relation names, attribute names, and attribute types. When the host simulator is initially invoked, any relation template descriptions stored in the "TEMPLATES" file are read into the "rel_templates" buffer. The capability to create additional templates is provided when the user selects the first option from the menu, as shown in Figure 2, and causes the template construction procedure, "create_template", to be invoked.

Initially, the template construction procedure allows existing relation templates to be displayed, one at a time, until the user desires to stop or until all templates have been shown. Then, the user is prompted to determine if a new relation template should be created. If none are to be created, the program returns to the main menu display. Otherwise, the user is prompted for a relation name. The user is then prompted for each attribute name and attribute type (character or integer) until the user desires to stop or until a designated attribute limit is reached. The new relation template is then stored into the "rel_templates" buffer. The buffer's contents will be saved into the "TEMPLATES" file when the host simulator program terminates, via the "QUIT" option on the main menu.

2.2. Displaying Relation Templates

The capability to display existing relation templates is provided when the user selects the second option from the main menu and causes the template displaying procedure, "show_templates", to be invoked. This procedure displays relation templates, one at a time, from the "rel_templates" buffer until the user desires to stop or until all templates have been shown. Afterwards, the program returns to the main menu display.

2.3. Creating a Request File

The capability to create a file of RRDS requests is provided when the user selects the third option from the main menu and causes the driver program, "create_req_file", of the request generating procedures to be invoked. The "create_req_file" procedure first prompts the user for the name of the request file to be created. The user is then given the option of either generating a set of RRDS requests himself or having the system generate a set of requests automatically.

2.3.1. Automatic Request Generation

The program currently generates only "INSERT" requests automatically. If a set of "INSERT" requests is to be created, the corresponding request generating procedure, "auto_insert", is invoked. The user is prompted for a relation name from a list of relation names. This list is derived from the relation templates stored in the "rel_templates" buffer. The user is then prompted for the number of "INSERT" requests to be generated automatically. These requests will be built from a set of data provided by the user. For each attribute of the selected relation, the user can give either a set of values or a range of values (a lower bound and an upper bound).

After the user has provided all of the necessary data values for each attribute, the program randomly selects attribute values from the attribute values provided and creates the "INSERT" requests according to the RRDS data manipulation language (DML). If a range is used to obtain values for an integer type attribute, a random value within the range will be generated for this attribute. The set of "INSERT" requests that are generated are then displayed to the user and stored into the request file. The procedure then returns to the main menu display.

2.3.2. Manual Request Generation

If RRDS requests are to be generated by the user, the request generating procedure, "man_req_gen", is invoked. This procedure displays a menu of RRDS request types that can be created as shown in Figure 3. Each request type has its own procedure to construct the request according to the RRDS DML specifications. Each request constructing procedure prompts the user for all necessary information, such as relation names, attribute values, and "WHERE" clause conjunctions. The type of information required by a request generating procedure depends on the type of RRDS request that it creates. Each request is stored into a request array as it is constructed. After the request has been created, it is displayed to the user. Then, the menu of RRDS request types is redisplayed to allow more requests to be created and included in the request file.

The above request generating procedures are menu driven, i.e., they prompt the user for individual values needed in a request. There is also a menu option that allows the user to enter an entire request. This ad hoc option invokes the "man_adhoc" procedure and allows the user, for example, to enter an incorrect request in order to test the parsing capability of the controller program.

When the "QUIT" option is selected from the menu, the newly created requests in the request array are written into the request file. Then, the main menu is redisplayed.

```
Please select a number to create the desired request.
[1] AVERAGE     [5] INSERT      [9]  MIN       [13] UNION
[2] COUNT       [6] INTSECT     [10] PROJECT   [14] UPDATE
[3] DELETE      [7] JOIN        [11] SELECT    [15] ADHOC
[4] DIFF        [8] MAX         [12] SUM       [0]  QUIT
->
```

Figure 3: Menu of RRDS Request Types

2.4. Displaying a Request File

The capability to display the contents of a request file is provided when the fourth option is selected from the main menu. This will cause the file displaying procedure, "show_req_file", to be invoked. The user is first prompted for the name of an existing request file. After providing the proper name, the requests in the file are stored into a request array. Then, the "display_req" procedure is invoked. The "display_req" procedure allows a user to view a specific request or all of the requests in the request array. The procedure pauses after every fifth request when all the requests are to be viewed (this will avoid screen scrolling before the user had a chance to read the requests). Requests can be displayed until the user decides to stop.

2.5. Editing a Request File

The capability to edit an existing request file is provided when the fifth option is selected from the main menu, causing the editing procedure ("edit_req_file") to be invoked. This procedure prompts the user for the name of an existing request file and then reads the requests into a request array. Then, a menu of editing options is displayed to the user. Requests can be added, deleted, displayed, or modified in a request file. However, all changes are first made in the request array. This will reduce the disk I/O and it will also provide the user the option of saving or not saving the changes into the request file before leaving the editor program. The editing options are described in the following sections.

2.5.1. Adding a Request

When the option to add a new request is selected from the editing menu, the request inserting procedure, "add_req", is invoked. First, a menu of request types that can be created for addition to the request file is displayed, similar to the one shown in Figure 3. After selecting the request type, the user is prompted for the position to add the request. After the position is chosen within the valid range, the request array is adjusted so that the request can be added at the desired location without overwriting an existing request. The appropriate procedure, depending on the request type that was chosen, is then invoked to generate the request interactively. The newly created request is stored in the request array and displayed to the user. Then, the editing menu is redisplayed to the user.

2.5.2. Deleting a Request

When the option to delete a request is chosen from the editing menu, the request deleting procedure, "delete_req", is invoked. The user is then prompted for the position of the request, in the request array, to be deleted. The request selected for deletion is displayed to the user to determine if it really is the request that should be deleted. If the user confirms that it should be deleted, the request array is adjusted so that all requests after the selected request are moved into the position of the preceding request. This causes the selected request to be overwritten in the request array. After the request is overwritten, the editing menu is redisplayed to the user.

2.5.3. Displaying a Request

When the option to display a request is chosen from the editing menu, the request displaying procedure, "display_req", is invoked. This procedure has been described in Section 2.4 and will not be repeated here. Requests can be displayed until the user decides to stop. Afterwards, the editing menu is redisplayed to the user.

2.5.4. Modifying A Request

When the option to modify a request is chosen from the editing menu, the request modifying procedure, "modify_req", is invoked. This procedure prompts the user for the position of the request in the request array that should be changed. The selected request is displayed to the user to determine if it is the correct request to be modified. If the user confirms that it is the correct request, the user is then prompted for the string that is to be changed. Then, the user is prompted for the replacement string. If the string to be changed exists in the request, it is replaced with the replacement string. However, if the replacement string is longer or shorter than the original string, the request is adjusted, accordingly, so that neither data will be overwritten nor extraneous data will remain in the request. The modified request is then displayed to the user followed by a redisplay of the editing menu.

2.5.5. Terminating the Editor

When the option to terminate the editing program is chosen from the editing menu, the editing loop is exited. Then, the user is prompted to determine if all the changes in the request array should be saved. If the user desires to save the changes, all requests in the request array are written into the request file from which they were read. Then, the editing program, "edit_req_file", terminates and the program returns to the main menu display.

2.6. Running Requests and Obtaining Results

The capability to send requests to the controller program for processing is provided when the sixth option is selected from the main menu. This will cause the request handling procedure, "run_req_file", to be invoked. If a socket channel has not been established with the controller program, an interprocess channel is first created using the socket facility commands. (If the controller program is not running, it must be invoked before the host program proceeds. The controller program, as described in Chapter 3, will establish a channel connection with the host so that requests can be received from the host program.)

After a socket channel has been established and both programs are attached to it, the "run_req_file" procedure will prompt the user for the name of a request file containing requests to be processed. After providing the name of an existing request file, requests are read from the file and stored into a request array to minimize disk I/O. Then, the user is prompted to determine if the request processing results should be logged in a file, displayed at the terminal, or both. If necessary, the user is prompted for the name of the output file to log the results. Then the user is prompted to determine if all of the requests or selected requests should be sent to the controller program for processing. Depending on the user's response, either the procedure to process all requests, "run_all", or the procedure to process selected requests, "run_selected", will be invoked. These procedures are described in the following sections.

2.6.1. Running All Requests in a File

When the procedure to process all requests, "run_all", is invoked, each request in the request array is passed to the controller program, one request at a time, on the interprocess channel using the "send" command of the socket facility. Then, the program awaits the request results from the controller program using the socket channel receiving command ("recv").

When a result arrives from the controller program, the result is stored into a reply buffer. Then, depending on the users's output direction, the request and its result will be logged in a file, displayed at the terminal, or both. However, if the result contains a continuation symbol, the procedure enters a loop to await further data from the controller program. Within this loop, data is sent to the proper output location(s) after it has been received on the interprocess channel via the "recv" command of the socket facility. The loop is terminated when a termination symbol is received from the controller program, which indicates the end of data transmission from the controller program. The "run_all" procedure then transmits the next request in the request array to the controller program using the "send" command of the socket facility.

2.6.2. Running Selected Requests

When the procedure to process selected requests, "run_selected", is invoked, the user is able to run requests from the file and new requests generated interactively at the terminal.

2.6.2.1. Running Selected Requests in a File

If the user wishes to run a request from the file, he is prompted for the position of the request in the request array. The selected request is shown to the user to determine if it is the correct request to be sent to the controller program. If the user agrees, the selected request is transmitted to the controller program via the "send" command of the socket facility. Then, the "run_selected" procedure awaits the request result from the controller program via the "recv" command of the socket facility.

The handling of the request result, when it arrives from the controller program, has been described in section 2.6.1 and will not be repeated here.

2.6.2.2. Running Selected Requests Generated from a Terminal

The "run_selected" procedure also allows a user to interactively generate a request at the terminal and transmit it to the controller program for processing. If a user wants to interactively generate a request at the terminal instead of selecting a request from the file, a menu of request types that can be created is displayed to the user as shown in Figure 3. When a request type is selected from the menu, the appropriate procedure is invoked to generate the request interactively. The newly created request is stored into a terminal request buffer. The request in the terminal request buffer is then transmitted to the controller program via the "send" command of the socket facility. The "run_selected" procedure then awaits the request result from the controller program via the "recv" command of the socket facility.

After receiving the result from the controller program, the "run_selected" procedure prompts the user for another request. As described before, the request can be from the file or generated from the terminal.

2.7. Program Termination

When the "QUIT" option is selected from the main menu of the host program, termination of the host simulator program and the RRDS controller program will result. First, the host program determines if an interprocess channel was established with the controller program. If a socket channel did exist, a termination symbol is sent to the controller program. As described in Chapter 3, when the controller program receives the termination symbol from the host program, it will exit the request processing loop, close the "BROADCAST" file, and it will terminate. Then, the host program removes the socket channel. Afterwards, the relation templates in the "rel_templates" buffer are stored into the "TEMPLATES" file. Finally, the host program closes the "TEMPLATES" file and terminates.

3. IMPLEMENTATION OF THE RRDS CONTROLLER

The RRDS controller program was implemented on the VAX 11/780 computer, operating under the Berkeley 4.3 version of UNIX, and was written in the C programming language [Kern78]. The program implements the controller's functions as described in Chapter 1. System tools, such as YACC [John79], LEX [Lesk79], and the socket facility [UNIX83], were also utilized to construct the controller program. The socket facility was used for interprocess communication with the host simulator program. Whereas, LEX and YACC were used to develop the parser for the RRDS data manipulation language (DML). The controller program receives requests from the host, parses them, passes the valid requests to simulated RCs, receives results from simulated RCs, coalesces the results, and forwards results to the host. These operations are described in the following sections.

3.1. Receiving a Request from the Host

An interprocess communication channel is established when the host simulator program initially sends a request to the controller for processing. When the controller is initially invoked, it establishes a connection to the socket channel and waits for the first request to be received via the communication socket. When a request is received, it is stored into a buffer.

3.2. Parsing the Request

Upon receiving a request, the LEX input pointer, "yyin", is set to the first character position of the request buffer. The request is then parsed by invoking the YACC generated parser "yyparse()". "yyparse", in turn, invokes the LEX lexical analyzer, "yylex()", to obtain tokens from the request. These tokens are matched by "yyparse()" according to the RRDS data manipulation language (DML). If a request's syntax is valid, "yyparse()" returns a value of "0", otherwise it returns a value of "1".

3.3. Rejecting Incorrect Requests

If the request is invalid because of a syntax error, the error handling procedure of the controller program, "bad_req", is invoked. This procedure writes an "INVALID REQUEST" message into the "reply" buffer that will be transmitted to the host.

3.4. Forwarding Correct Requests to the RCs

When a request is valid, it is written to the "BROADCAST" file. The "BROADCAST" file is used to simulate broadcasting the request to the RCs since the RCs and DBMS software are not yet implemented.

3.5. Simulating RC Result Data and Coalescing It

In order to implement and test the controller's coalescing capabilities, result data files were created to simulate the output data that could be generated by operational RCs. A procedure, "get_results", was written that analyzes the request type and obtains result data from the RC data files based on the request type. For requests such as "INSERT" and "DELETE", a simple request completion message is stored into the "reply" buffer, which will be forwarded to the host. For aggregate requests, such as "MAX" and "SUM", the first data item is read from each result data file and coalesced by the corresponding procedure, such as "co_MAX" and "co_SUM". After deriving the proper aggregate value, it is written into the "reply" buffer.

For requests that return relation fragments, such as "SELECT" and "PROJECT", the "co_RECORDS" procedure is invoked. This will cause a random amount of records to be read from each result data file and stored into an array. Then, the array is coalesced to remove duplicate records, which simulates duplicate relation fragments. Upon completion, a continuation symbol is written into the "reply" buffer. This will notify the host that more data follows this message when the results are sent to it. The records in the coalesced array will then be forwarded to the host.

3.6. Forwarding Request Results to the Host

The controller forwards request results to the host by invoking the "send_results" procedure. This procedure uses the "send" command of the socket facility to transmit data, on the interprocess channel, to the host. Initially, the "send_results" procedure transmits the contents of the "reply" buffer, which was appropriately set in either the valid request procedure ("get_results") or the invalid request procedure ("bad_req"). However, if the reply buffer contains the continuation symbol, the "send_results" procedure will also transmit the coalesced relation fragments, one record at a time, to the host. After sending the last record, a termination symbol is sent to the host to signal the end of data transmission.

After sending the results, the controller waits for the next request via the "recv" command of the socket facility.

3.7. Program Termination

When the controller program receives the termination symbol as a request, the controller program will exit its processing loop, close the "BROADCAST" file, and then terminate.

4. SUMMARY AND CONCLUSIONS

In this paper, we have presented the design and implementation of two software systems for handling RRDS requests. A host simulator system was constructed to generate RRDS requests similar to those that could be produced by an operational host computer using RRDS. The host simulator is menu driven and enables a user to easily generate, edit, and forward RRDS requests to the RRDS controller, which was also developed.

The RRDS controller system accepts RRDS requests from the host simulator via a duplex socket channel established for interprocess communication. The requests are then parsed, according to the RRDS data manipulation language, by the parser developed using the "LEX" and "YACC" facilities of UNIX. Valid requests are then sent to the "BROADCAST" file, to simulate transmitting a request to the RCs, since the broadcast bus, replicated computers (RCs), and DBMS software have not been implemented. Because the RCs are not running, result data files are used to simulate and generate RC result data.

Result data is generated, based on the request type, to produce either a reply, relation fragments, or aggregate values. The results from RCs are used to verify the coalescing operation of the controller program. Relation fragments are coalesced to remove duplicates and aggregate values are combined to produce the final aggregate value. The results for valid requests are sent to the host program via the socket channel. For invalid requests, an error message is sent to the host.

As operational components, such as the RCs and host computer, are developed and linked to the RRDS controller, the bus simulation interfaces, between the controller and the host or between the controller and RCs, can be replaced in the controller program. The "send" and "recv" interfaces of the socket channel, used between the host simulator and controller programs, can be replaced by corresponding bus I/O interfaces. Also, the file write commands to the "BROADCAST" file and the file read commands from the result data files can be replaced by corresponding bus I/O interfaces between the RCs and RRDS controller program.

REFERENCES

[Hans86a] Hanson, J.G., "Design and Performance Analysis of a Relational Replicated Database System," Doctoral Research Proposal, Department of Computer Science, University of Central Florida, Orlando, Florida, December 1986.

[Hans86b] Hanson, J.G., and Orooji, A., "Query Processing in a Relational Replicated Database System (RRDS)," Unpublished Technical Report.

[Hans86c] Hanson, J.G., and Orooji, A., "A Data Manipulation Language for a Relational Replicated Database System (RRDS)," Unpublished Technical Report.

[Hans87] Hanson, J.G., and Orooji, A., "A Taxonomy of Database Systems," Technical Report CS-TR-87-10, University of Central Florida, Orlando, Florida, June 1987.

[Hara83] Haran, B., and DeWitt, D.J., "Database Machines: An Idea Whose Time has Passed? A Critique of the Future of Database Machines," pp. 166-187, Database Machines, Edited by Leilich, H.O., and Missikoff, M., Springer-Verlag, 1983.

[Jack75] Jackson, M.A., "Principles of Program Design," Academic Press, 1975.

[John79] Johnson, S.C., "YACC: Yet Another Compiler-Complier," UNIX Time-Sharing System: UNIX Programmer's Manual, Bell Telephone Laboratories, Incorporated, Murray Hill, N.J., 1979.

[Kern78] Kernighan, B.W., and Ritchie, D.M., "The C Programming Language," Prentice-Hall, 1978.

[Lesk79] Lesk, M.E., and Schmidt, E., "Lex - A Lexical Analyzer Generator," UNIX Time-Sharing System: Unix Programmer's Manual, Bell Telephone Laboratories, Incorporated, Murray Hill, N.J., 1979.

[UNIX83] UNIX Programmer's Manual 4.2 BSD, Bell Telephone Laboratories, Incorporated, 1983.

APPENDIX A

RRDS DML SPECIFICATION

In this appendix the formal BNF syntax for the RRDS data manipulation language (DML) is presented (square brackets [ ] indicate optional constructs).

```
<Query>          ::= <Select-Query> | <Project-Query> |
                     <Insert-Query> | <Delete-Query> |
                     <Update-Query> | <Join-Query> |
                     <Union-Query> | <Diff-Query> |
                     <Intsect-Query> | <Agg-Query>

<Agg-Query>      ::= <Count-Agg> | <Sum-Agg> | <Min-Agg> |
                     <Max-Agg> | <Ave-Agg>

<Select-Query>   ::= SELECT [ALL] FROM <Target-Relation>
                     [WHERE
                       (<Specifier>)]

<Project-Query>  ::= PROJECT (<Attr-List>) FROM <Target-Relation>
                     [WHERE
                       (<Specifier>)]

<Insert-Query>   ::= INSERT (<Record>) INTO <Target-Relation>

<Delete-Query>   ::= DELETE [ALL] FROM <Target-Relation>
                     [WHERE
                       (<Specifier>)]

<Update-Query>   ::= UPDATE (<Modifier>) IN <Target-Relation>
                     [WHERE
                       (<Specifier>)]

<Union-Query>    ::= <Target-Relation> UNION <Target-Relation>

<Diff-Query>     ::= <Target-Relation> DIFF <Target-Relation>

<Intsect-Query>  ::= <Target-Relation> INTSECT <Target-Relation>

<Join-Query>     ::= <Target-Relation> JOIN <Target-Relation>
```

```
<Count-Agg>       ::= COUNT FROM <Target-Relation>
                      [WHERE (<Specifier>)]

<Sum-Agg>         ::= SUM (<Attribute>) FROM <Target-Relation>
                      [WHERE (<Specifier>)]

<Min-Agg>         ::= MIN (<Attribute>) FROM <Target-Relation>
                      [WHERE (<Specifier>)]
<Max-Agg>         ::= MAX (<Attribute>) FROM <Target-Relation>
                      [WHERE (<Specifier>)]

<Ave-Agg>         ::= AVERAGE (<Attribute>) FROM <Target-Relation>
                      [WHERE (<Specifier>)]

<Target-Relation> ::= <String>

<Record>          ::= <Record-Segment>

<Record-Segment> ::= <Attr-Value-Pair> |
                       <Attr-Value-Pair>,<Record-Segment>

<Attr-Value-Pair> ::= <Attribute> <Assign-Op> <Value>

<Attr-List>       ::= <Attribute> | <Attribute>,<Attr-List>

<Attribute>       ::= <String>

<Specifier>       ::= <Conjunction> | <Conjunction> OR <Specifier>

<Conjunction>     ::= (<Pred-List>)

<Pred-List>       ::= <Predicate> | <Predicate> AND <Pred-List>

<Predicate>       ::= <Attribute> <Rel-Op> <Value>

<Modifier>        ::= <Assign-Statement-List>

<Assign-Statement-List>
                  ::= <Assign-Statement> |
                      <Assign-Statement>,<Assign-Statement-List>

<Assign-Statement> ::= <Attribute> <Assign-Op> <Arithm-Expression>

<Arithm-Expression> ::= <Term> |
                          <Arithm-Expression> <Add-Op> <Term>

<Term>            ::= <Factor> | <Term> <Mul-Op> <Factor>

<Factor>          ::= <Attribute> | <Value> | (<Arithm-Expression>)

<Value>           ::= <String> | <Number>

<String>          ::= <Uc-Letter> | <Uc-Letter> <Alph-Num>
```

```
<Alph-Num>       ::= <Letter> | <Digit> |
                     <Letter> <Alph-Num> | <Digit> <Alph-Num>

<Number>         ::= <Integer> | <Real> |
                     <Add-Op> <Integer> | <Add-Op> <Real>

<Real>           ::= <Integer>.<Integer>

<Integer>        ::= <Digit> | <Digit> <Integer>

<Letter>         ::= <Uc-Letter> | <Lc-Letter>

<Uc-Letter>      ::= A | B | C | ... | Z

<Lc-Letter>      ::= a | b | c | ... | z

<Digit>          ::= Ø | 1 | 2 | ... | 9

<Add-Op>         ::= + | -

<Mul-Op>         ::= * | /

<Rel-Op>         ::= < | <= | > | >= | <> | =

<Assign-Op>      ::= :=
```

APPENDIX B

PROGRAM DESIGN LANGUAGE SPECIFICATION AND JACKSON CHART DESIGN SPECIFICATION

In this appendix, the design tools used for the RRDS design documents are presented. Section 1 describes the program design language. Section 2 describes Jackson Chart constructs and usage.

B.1. Program Design Language Specification

The program design language provides constructs such as conditional statements, repetitive statements, and subroutines. These constructs are described in the following sections.

B.1.1. Comments

A design document can include comments by enclosing each comment within the delimiters "/*" and "*/".

Example: /* This is a comment */

B.1.2. Assignment Statement

An assignment statement is used to assign a value to a variable, where the variable on the left side of the "=" symbol is assigned the value of the expression on the right side of the "=" symbol. An expression can be an arithmetic clause or a descriptive clause. Each assignment statement is terminated with a semicolon.

```
   Format:  variable = expression;
   Examples:  count = number of predicates in specifier;
              count = count + 1;
```

B.1.3. Conditional Statements

A conditional statement provides a way to control the operational flow of a program based on the result of a condition. The design language provides a simple "if" statement, a compound "if-then-else" statement, and a "case" statement as conditional constructs. Each construct contains a conditional expression that is used to determine if the operations described within the construct should be performed. However, the case construct uses the value of a conditional expression to determine the set of operations to perform. The format and examples for each of these conditional constructs are shown below.

```
     Format:  if expression then
                 statement
              end if
     Example:
              if the request is valid then
                 send the request to the RCs
              end if
```

```
Format:  if expression then
            statement1
         else
            statement2
         end if
Example:
         if #of_pred > MAX_PRED then
            print an error message
         else
            find record addresses for each predicate
         end if
Format:  case expression
              value1:
                 statement1
              value2:
                 statement2
              value3:
                 statement3
              ...
              otherwise:
                 statementn
         end case
```

```
Example:  case request type
               INSERT:
                  call insert_processing();
               RETRIEVE:
                  call retrieve_processing();
               otherwise:
                  print an error message
          end case
```

B.1.4. Repetitive Statements

These type of statements provide a way to repeat a set of operations. The number of iterations is specified in the expression clause of the construct. The language provides a "for" statement and a "while" statement. The format and examples for these constructs are shown below.

```
Format:  for expression
             statement
         end for
Examples:  for each element a in s
               total = total + a;
           end for
           for k = 10 to 50 by 5
               print k
           end for
Format:  while expression
             statement
         end while
```

```
Examples:  while there are more requests
                process the request
           end while
           while a > b
                a = a - b;
           end while
```

B.1.5. Input/Output Statements

The language provides statements to read and write data from a file or a terminal, based on a "f_" or "t_" prefix. Each statement contains an expression clause to describe the data being read/written. The format and examples for these I/O statements are shown below.

```
     Format:  f_read  expression
              f_write expression
              t_read  expression
              t_write expression
    Examples:
        f_read  requests in the request file into the request array
        f_write requests in the request array into the request file
        t_read  user selection
        t_write requests in the request array 10 at a time
```

B.1.6. Subroutines

The language provides a way to describe procedures and functions used by a program. Procedures are described using the "proc" construct. Whereas, functions are described with the "func" construct. A "proc" construct is invoked with a "call" statement and a "func" construct is implicitly invoked when it is used in an expression. The format of these constructs, the invoking statements, and examples are shown below.

```
Format:
proc pname (input: variables, input/output: variables
                                    output: variables);
     statement
end proc pname;
Example:
proc sort (input: n, input/output: tbl, output: nothing);
           /* This procedure sorts the table "tbl" */
           /* using Quick Sort                     */
end proc sort;
Format to invoke a "proc" construct:
   call pname(variables);
Example:  call sort(count, a);
Format to return from the middle of a "proc" construct:
   return;
Format:
func fname (input: variables) returns (type);
     statement
     return(value);
end func fname;
```

```
Example:
func factor (input: n) returns (integer);
    /* This function returns the factorial of n */
end func factor;
Format to invoke a function:
   variable = fname(variables);
Example:  f = factor(6);
```

B.2. Jackson Chart Specification

Jackson Charts [Jack75] are used to model the structure of a program. Using Jackson Charts, a program is subdivided into a hierarchical structure that can consist of both elementary and composite logic components. The elementary component is related to a single operation and is not further subdivided. Whereas, a composite logic component binds various lower level components, such as elementary and logic components.

Components are represented as labeled blocks and are placed in operational order from left to right. Component blocks also contain symbols to describe their execution within the program. These symbols denote the sequence, selection, and iteration constructs used in a Jackson Chart. These three constructs are described in the following sections.

B.2.1. The Sequence Construct

The absence of a symbol, for a set of component blocks at the same level, indicates that the blocks will be executed in sequence from left to right. An example of a sequence construct is shown in Figure 4.a. In this example, block "A" consists of block "B", followed by block "C", followed by block "D".

B.2.2. The Selection Construct

A "o" symbol, in the upper right corner of a set of component blocks at the same level, indicates that only one of these blocks will be executed. An example of a selection construct is shown in Figure 4.b. In the example, block "A" consists of either block "B", block "C", or block "D".

B.2.3. The Iteration Construct

A "*" symbol in the upper right corner of a component block indicates that the block can be executed zero or more times. An example of an iteration construct is shown in Figure 4.c. In this example, block "A" consists of block "B" zero or more times in succession.

B.3. A Sample Program

An example of a program structure, represented in Jackson Chart, is shown in Figure 5. The corresponding program design, written in the design language, is shown in Figure 6.

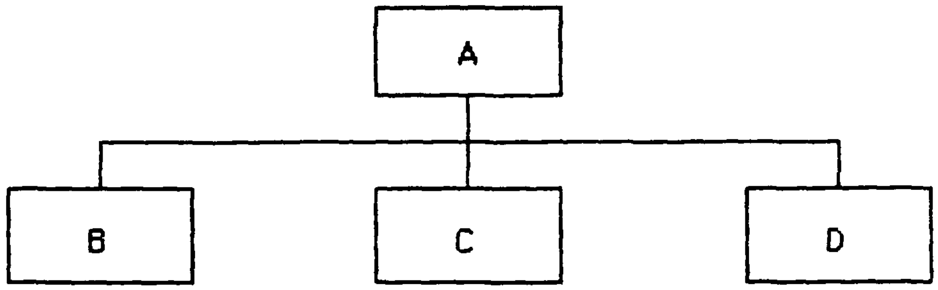


Figure 4.a: Example Of A Sequence Construct

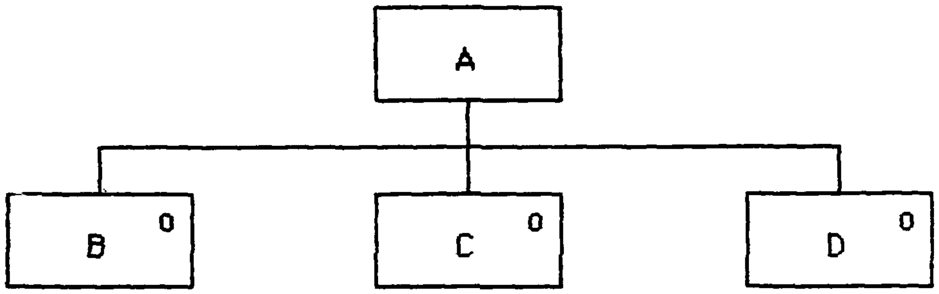


Figure 4.b: Example Of A Selection Construct

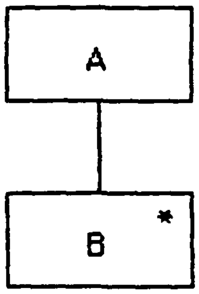


Figure 4.c: Example Of An Iteration Construct

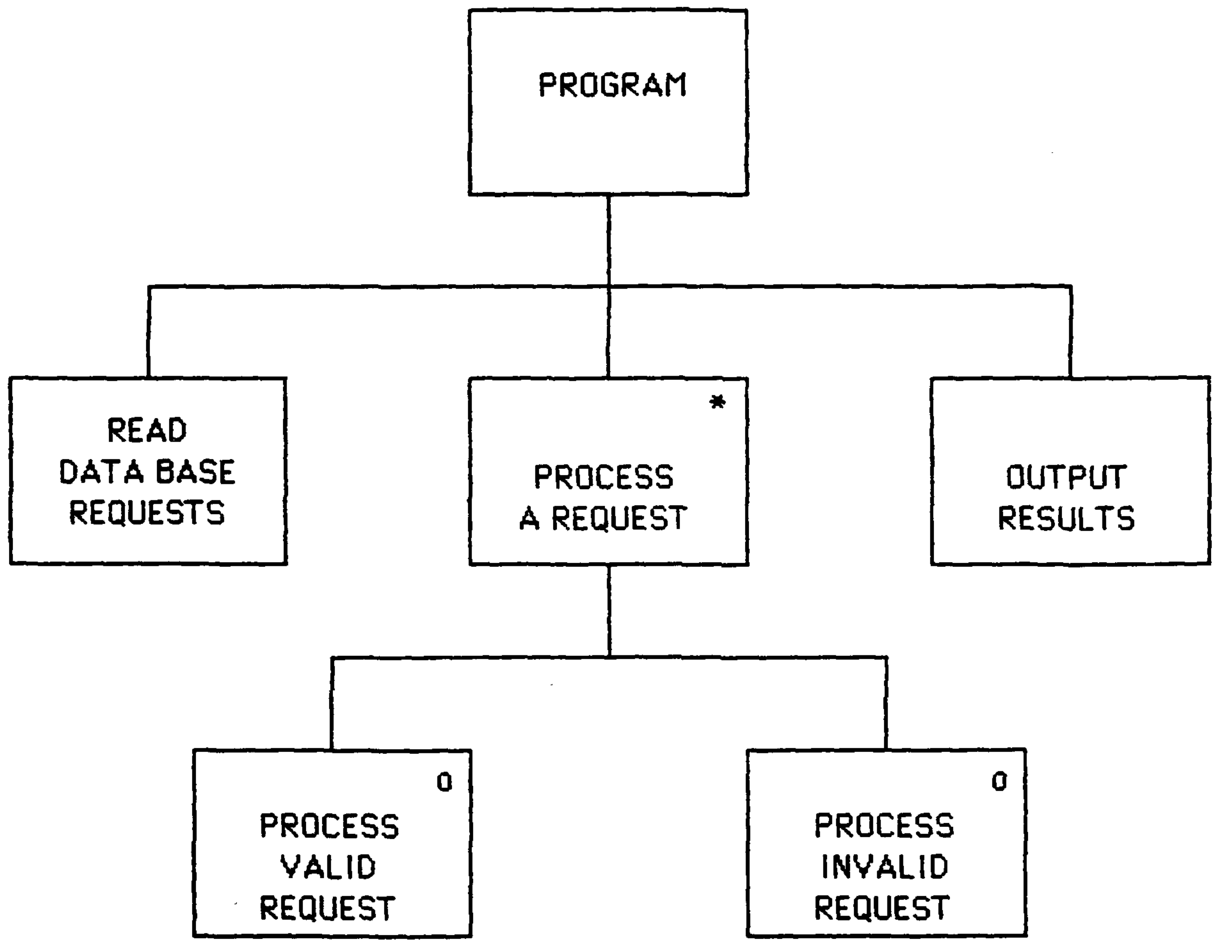


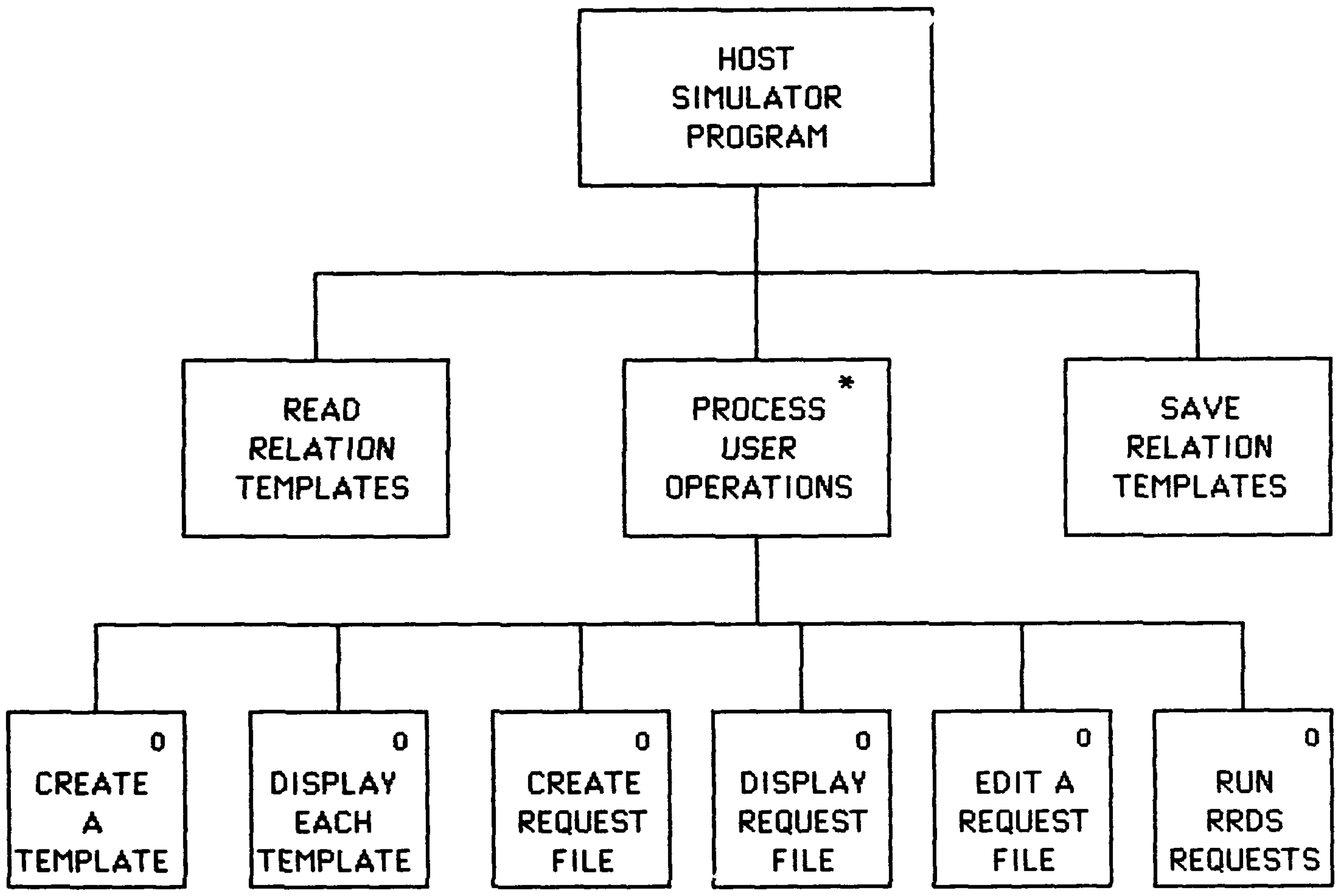
Figure 5: Sample Program Defined Using A Jackson Chart

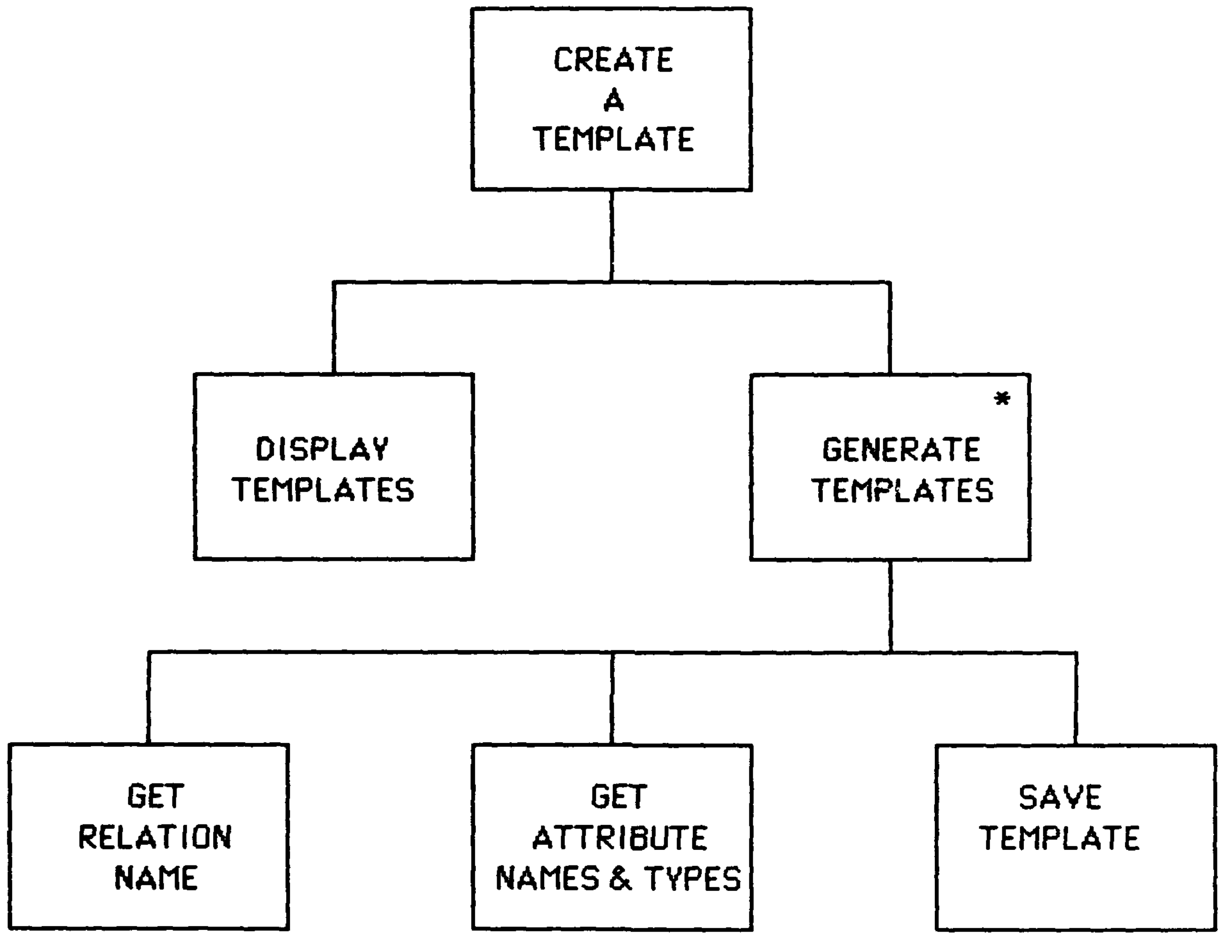
```
proc request_processing();
     f_read the data base requests from request file into an array
     while there are more requests in the array
           t_write the next request in the array
           valid = value from request parser;
           if valid = TRUE then
              call process_valid_request();
           else
              call process_invalid_request();
           end if
           t_write the request results
     end while
end proc request_processing;
```

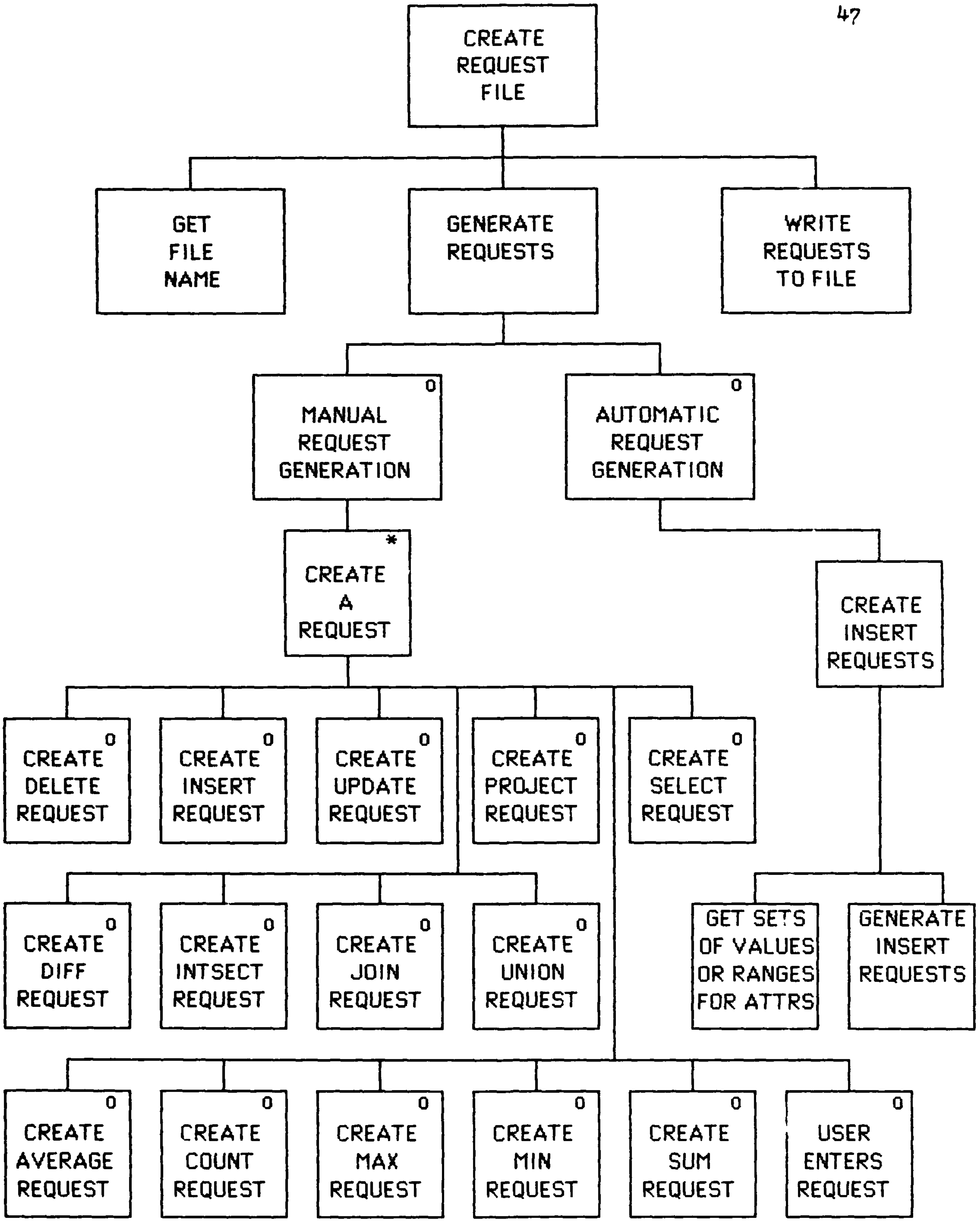
Figure 6: Sample Program Defined in the Design Language

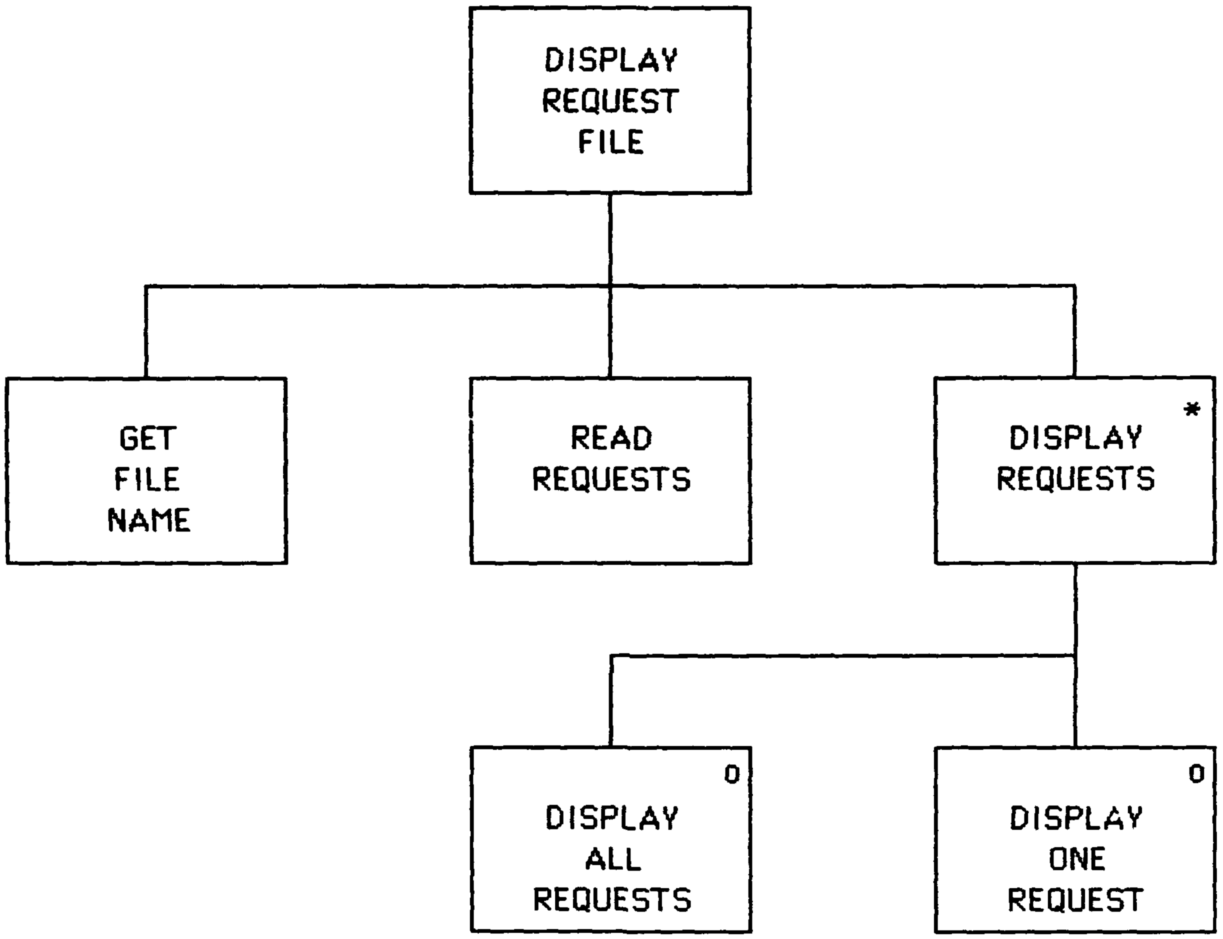
APPENDIX C

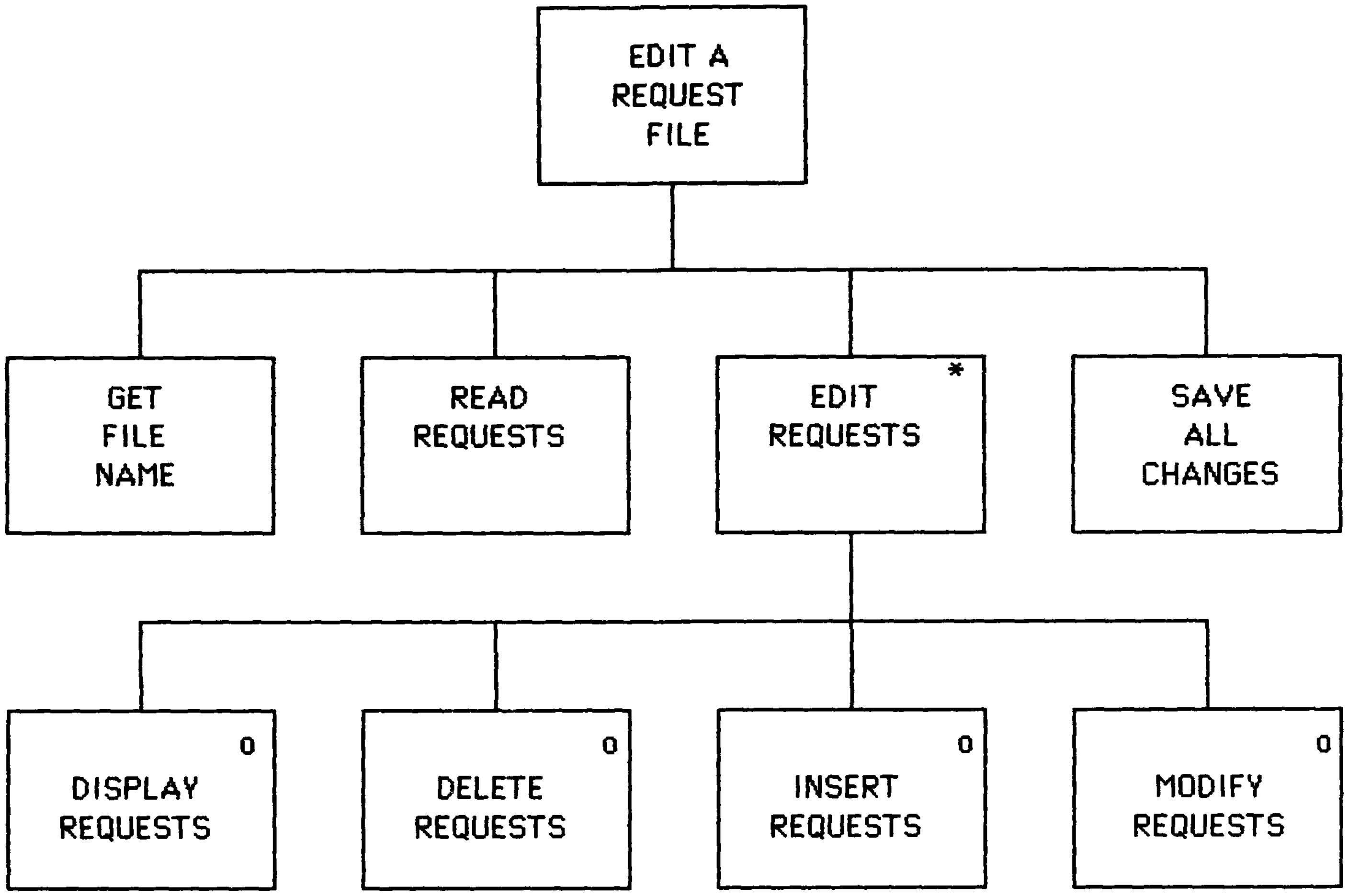
JACKSON CHARTS AND PROGRAM DESIGN LANGUAGE FOR THE HOST SIMULATOR PROGRAM

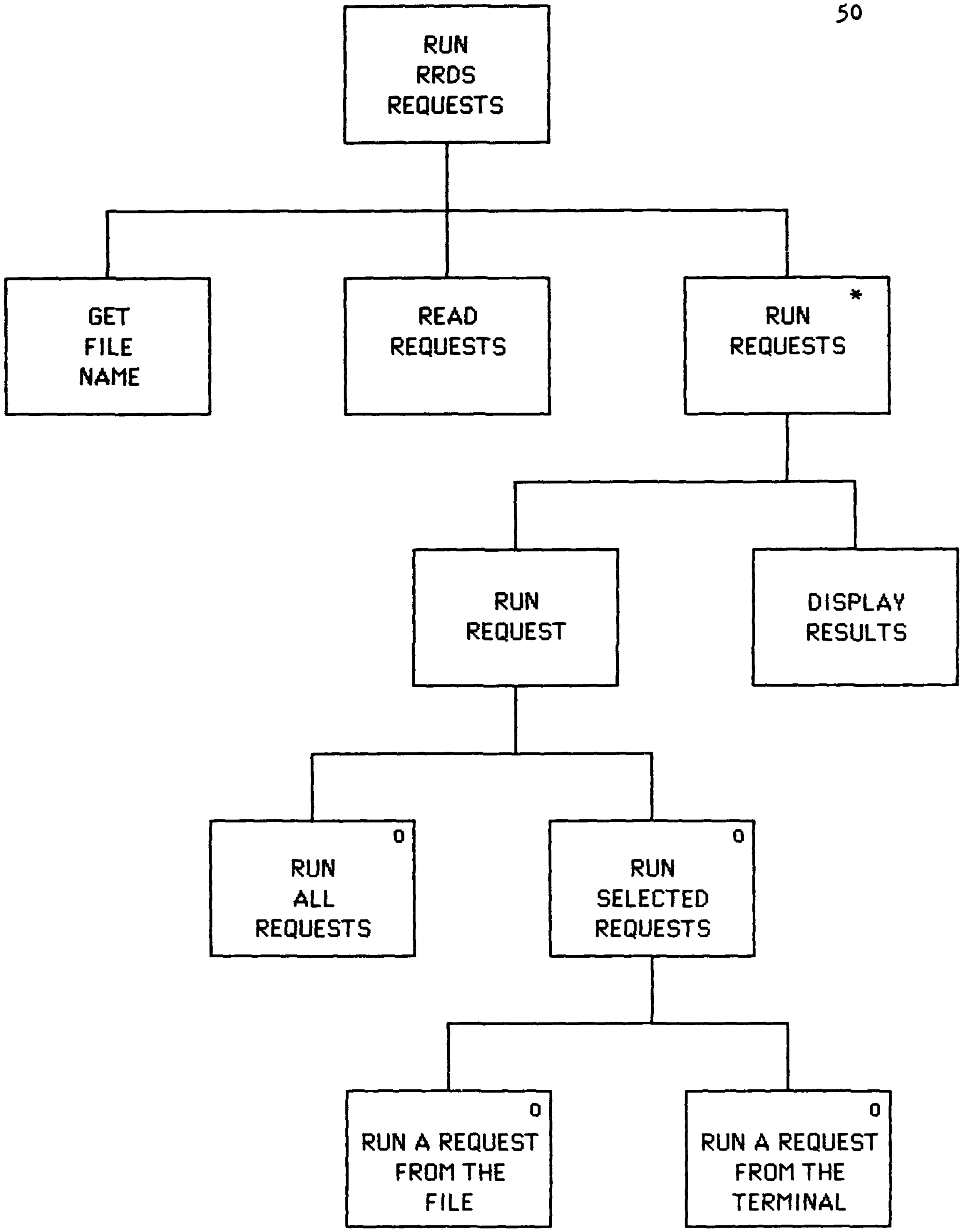

HOST SIMULATOR PROGRAM
READ RELATION TEMPLATES
PROCESS USER OPERATIONS *
SAVE RELATION TEMPLATES
CREATE A TEMPLATE 0
DISPLAY EACH TEMPLATE 0
CREATE REQUEST FILE 0
DISPLAY REQUEST FILE 0
EDIT A REQUEST FILE 0
RUN RRDS REQUESTS 0


CREATE A TEMPLATE
DISPLAY TEMPLATES
GENERATE TEMPLATES
*
GET RELATION NAME
GET ATTRIBUTE NAMES & TYPES
SAVE TEMPLATE


CREATE REQUEST FILE
GET FILE NAME
GENERATE REQUESTS
WRITE REQUESTS TO FILE
MANUAL REQUEST GENERATION O
AUTOMATIC REQUEST GENERATION O
CREATE A REQUEST *
CREATE INSERT REQUESTS
CREATE DELETE REQUEST O
CREATE INSERT REQUEST O
CREATE UPDATE REQUEST O
CREATE PROJECT REQUEST O
CREATE SELECT REQUEST O
CREATE DIFF REQUEST O
CREATE INTSECT REQUEST O
CREATE JOIN REQUEST O
CREATE UNION REQUEST O
GET SETS OF VALUES OR RANGES FOR ATTRS
GENERATE INSERT REQUESTS
CREATE AVERAGE REQUEST O
CREATE COUNT REQUEST O
CREATE MAX REQUEST O
CREATE MIN REQUEST O
CREATE SUM REQUEST O
USER ENTERS REQUEST O


DISPLAY REQUEST FILE
GET FILE NAME
READ REQUESTS
DISPLAY REQUESTS *
DISPLAY ALL REQUESTS 0
DISPLAY ONE REQUEST 0


EDIT A REQUEST FILE
GET FILE NAME
READ REQUESTS
EDIT REQUESTS *
SAVE ALL CHANGES
DISPLAY REQUESTS O
DELETE REQUESTS O
INSERT REQUESTS O
MODIFY REQUESTS O


RUN RRDS REQUESTS
GET FILE NAME
READ REQUESTS
RUN REQUESTS *
RUN REQUEST
DISPLAY RESULTS
RUN ALL REQUESTS 0
RUN SELECTED REQUESTS 0
RUN A REQUEST FROM THE FILE 0
RUN A REQUEST FROM THE TERMINAL 0

```
/***********************************************************************/
/*                                                                     */
/*  This host simulator  program was developed and written by          */
/*  Leonard A. Lyon during the Spring 1987 Semester.  This host        */
/*  simulator program is designed for use with the RRDS Controller.    */
/*  This program is menu driven and allows users to interactively      */
/*  create, edit, and run RRDS requests.                               */
/*                                                                     */
/***********************************************************************/
/*             GLOBAL  VARIABLES             */
/*                                           */
/* CONTINUE             indicates more data follows                  */
/* TERMINATE            indicates end of data transmission           */
/* output               directed output(screen,file,both,S/F/B) */
/* recv_request         Controller's current request buffer          */
/* req_cnt              number of requests read from file            */
/* reply                Controller's reply to a request              */
/* tmpl_cnt             number of templates read from file           */
/* req_buffer[MAX_REQ]  array of requests                            */
/*                                                                   */
/*  Structure variables for relation templates                       */
/*                                                                   */
/* 1 attribute                                                       */
/*   2 at_name                                                       */
/*   2 at_type                                                       */
/*                                                                   */
/* 1 relation                                                        */
/*   2 rel_name                                                      */
/*   2 rel_nof_attrs                                                 */
/*   2 attribute [MAX_ATTRS]                                         */
/*                                                                   */
/* 1 rel_templates                                                   */
/*   2 relation [MAX_RELS]                                           */
```

```
proc main();
/* BEGIN HOST SIMULATOR */

     /* Initialize the string constants */

     BLANK       = " ";
     BOR         = "[";
     CONTINUE    = "+";
     EOR         = "]";
     TERMINATE   = "@";
     REL[0].OP   = "<=";
     REL[1].OP   = ">=";
     REL[2].OP   = "<>";
     REL[3].OP   = ">";
     REL[4].OP   = "<";
     REL[5].OP   = "=";

     /* Retrieve relation templates from TEMPLATES file */

     f_read relation templates from TEMPLATES file
                                  into rel_templates buffer
     while the user wants to perform more operations
           t_write menu of host operations
           t_read desired operation to be performed
           case the desired operation
                CREATE A TEMPLATE:
                    call create_template();
                DISPLAY TEMPLATES:
                    call show_templates();
                CREATE REQUEST FILE:
                    call create_req_file();
                DISPLAY REQUEST FILE:
                    call show_req_file();
                EDIT REQUEST FILE:
                    call edit_req_file();
                RUN REQUEST FILE:
                    call run_req_file();
                QUIT:
                    /* exit the loop */
                OTHERWISE:
                    t_write error message
           end case
     end while

     /* Save relation template descriptions */

     if relation templates exist then
        f_write relation templates from rel_templates buffer
                                      into TEMPLATES
     end if
end proc main;

/*******************************************************************/
```

```
proc create_template();
/* This procedure displays existing relation templates and allows */
/* the user to create new relation templates.                     */
/* GLOBAL:  tmpl_cnt, rel_templates[]                             */

     /* First display existing relation templates */

     t_write templates one at a time

     /* Create the name of the new relation template */

     t_write Create another relation template ?
     t_read  user response
     while the user wants to create more templates and tmpl_cnt < MAX_RELS
         t_write Enter name of relation template
         t_read template name into rel_templates buffer

         /* Create the attribute names for the new relation template */

         while the user wants to add more attributes and < MAX_ATTRS
             t_write Enter name of attribute
             t_read attribute name into rel_templates buffer
             t_write Enter attribute type (C/I)
             t_read attribute type into rel_templates buffer
         end while
         t_write Create another relation template ?
         t_read user response
     end while
end proc create_template;

/*******************************************************************/
```

```
proc create_req_file();
/* This procedure allows the user to store rrds requests that are  */
/* created either manually or automatically by the system.         */
/* GLOBAL: req_cnt, req_buffer[]                                   */

     t_write Enter the name of the request file to be created
     t_read file name
     t_write Enter 'a' or 'm' for automatic or manual request generation
     t_read user response

     if response = a then
        call auto_req_gen();
     else if response = m then
        call man_req_gen();
     end if

     /* Save all created requests */

     f_write requests into request file from request array
end proc create_req_file;

/*******************************************************************/

proc auto_req_gen();
/* This procedure generates a set of RRDS requests automatically. */
/* (Only INSERT requests are currently generated automatically.)  */

     while the user wants to create more requests
           t_write request menu (INSERT, QUIT)
           t_read user choice
           case user choice
                INSERT:
                    call auto_insert();
                QUIT:
                    /* exit the loop */
                OTHERWISE:
                    t_write error message
           end case
     end while
end proc auto_req_gen;

/***********************************************************************/
```

```
proc auto_insert();
/* This procedure creates a set of rrds INSERT requests automatically */
/* GLOBAL: tmpl_cnt, rel_templates[], req_cnt, req_buffer[]            */

     t_write Generate INSERT requests from which relation ?
     rel_index = get_rel();

     /* Determine how many requests to create within limits */

     t_write How many requests do you want to create ? ( <= limit)
     t_read  number into req_bound

     /* Determine total values the user will provide per attribute */

     for each attribute of the relation template
         t_write How many data values do you want to provide ?
         if attribute type = integer then
            t_write  Enter Ø for range OR number < limit
         end if
         t_read number into input_bound array
     end for

     /* Get data for the attributes used in the request */

     for each attribute in the relation template
         if input_bound for this attribute != Ø then
            t_write Enter data at each prompt to create records
            t_write NOTE:
                    Character data must start with an upper case letter
            for index = 1 to input bound for this attribute
                  t_write attribute name
                  t_write attribute type
                  t_read attribute data
            end for
         end if

         if attribute type = integer and input_bound = Ø then
            t_write Enter a numeric lower bound
            t_read  number into lower bound array
            t_write Enter a numeric upper bound
            t_read number into upper bound array
         end if
     end for
     /* Now create the insert requests from the data provided */

     gen_index = 0;
     while gen_index < req_bound and req_cnt < request limit
          /* Include INSERT clause in the request */
          /* Select  a random attribute value */
          /* Include attribute value pairs in the request */
          /* Include the INTO clause in the request */
          /* Include the relation name in the request */
          gen_index = gen_index + 1;
     end while
end proc auto_insert;

/*********************************************************************/

proc man_req_gen();
/* This procedure provides a menu of RRDS request types that can be*/
/* created by the user.  The appropriate request generator program */
/* is executed based on user input.                                */
/* GLOBAL: req_cnt, req_buffer[]                                   */

     while the user wants to generate more requests
                          and is below the limit
           t_write a menu of RRDS request types
           t_read selected request type from menu
           case request type
                AVERAGE:
                   call man_average(req_buffer[req_cnt].request);
                   req_cnt =  req_cnt + 1;
                COUNT:
                   call man_count(req_buffer[req_cnt].request);
                   req_cnt =  req_cnt + 1;
                DELETE:
                   call man_delete(req_buffer[req_cnt].request);
                   req_cnt =  req_cnt + 1;
                DIFF:
                   call  man_diff(req_buffer[req_cnt].request);
                   req_cnt = req_cnt + 1;
                INSERT:
                   call man_insert(req_buffer[req_cnt].request);
                   req_cnt = req_cnt + 1;
                INTSECT:
                   call man_intsect(req_buffer[req_cnt].request);
                   req_cnt = req_cnt + 1;
                JOIN:
                   call man_join(req_buffer[req_cnt].request);
                   req_cnt =  req_cnt + 1;
                MAX:
                   call man_max(req_buffer[req_cnt].request);
                   req_cnt = req_cnt + 1;
                MIN:
                   call man_min(req_buffer[req_cnt].request);
                   req_cnt = req_cnt + 1;
                PROJECT:
                   call man_project(req_buffer[req_cnt].request);
                   req_cnt = req_cnt + 1;
                SELECT:
                   call man_select(req_buffer[req_cnt].request);
                   req_cnt = req_cnt + 1;
                SUM:
                   call man_sum(req_buffer[req_cnt].request);
                   req_cnt = req_cnt + 1;
                UNION:
                   call man_union(req_buffer[req_cnt].request);
                   req_cnt = req_cnt + 1;
                UPDATE:
                   call man_update(req_buffer[req_cnt].request);
                   req_cnt = req_cnt + 1;
                ADHOC:
                   call man_adhoc(req_buffer[req_cnt].request);
                   req_cnt = req_cnt + 1;
                QUIT:
                   /* exit the loop */
                OTHERWISE:
                   t_write error message
           end case
           t_write the newly created request
     end while
end proc man_req_gen;

/*********************************************************************/
```

```
proc man_adhoc(output: request);
/* This procedure allows a user to enter any type of request with   */
/* delimiters This routine is useful for creating erroneous rrds    */
/* requests to verify the syntax analysis capabilities of the RRDS  */
/* Controller.                                                       */

     t_write Enter request within delimiters
     t_read  request
end proc man_adhoc;

/*******************************************************************/

proc man_average(output: request)
/* This procedure creates a rrds AVERAGE  request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                                 */

     /* Get relation name for the request */

     t_write Generate AVERAGE request from which relation ?
     rel_index = get_rel();

     /* Get the attribute used in the request */

     t_write Generate AVERAGE request for which attribute ?
     t_write list of integer type attribute names
                  from the relation template
     t_read  the attribute name

     /* Get the WHERE clause if necessary */

     call get_where_clause(rel_index, where);

    /* Now create the AVERAGE request from the data provided */

       /* Include the AVERAGE clause                     */
       /* Include the selected attribute in the request */
       /* Include the FROM clause in the request        */
       /* Include the relation name in the request      */
       /* Include any WHERE clause in the request       */

end proc man_average;

/*****************************************************************/
```

```
proc man_max(output: request);
/* This procedure creates a rrds MAX  request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates,                             */

     /* Get relation name for the request */

     t_write Generate MAX request from which relation ?
     rel_index = get_rel();

     /* Get the attribute used in the request */

     t_write Generate MAX request for which attribute ?
     t_write list of attribute names from the relation template
     t_read the attribute name

     /* Get the WHERE clause if necessary */

     call get_where_clause(rel_index, where);

     /* Now create the MAX request from the data provided */

        /* Include the MAX clause                          */
        /* Include the selected attribute in the request */
        /* Include the FROM clause in the request          */
        /* Include the relation name in the request        */
        /* Include any WHERE clause in the request         */

end proc man_max;

/****************************************************************/
```

```
proc man_min(output: request);
/* This procedure creates a rrds MIN request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates,                            */

     /* Get relation name for the request */

     t_write Generate MIN request from which relation ?
     rel_index = get_rel();

     /* Get the attribute used in the request */

     t_write  Generate MIN request for which attribute ?
     t_write list of attribute names from the relation template
     t_read the attribute name

     /* Get the WHERE clause if necessary */

     call  get_where_clause(rel_index, where);

     /* Now create the MIN request from the data provided */

        /* Include the MIN clause in the request          */
        /* Include the selected attribute in the request */
        /* Include the FROM clause in the request         */
        /* Include the relation name in the request       */
        /* Include any WHERE clause in the request        */

end proc man_min;

/****************************************************************/
```

```
proc man_sum(output: request)
/* This procedure creates a rrds SUM request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates,                            */

     /* Get relation name for the request */

     t_write Generate SUM request from which relation ?
     rel_index = get_rel();

     /* Get the attribute used in the request */

     t_write Generate SUM request for which attribute ?
     t_write list of integer type attribute names
                  from the relation template
     t_read attribute name

     /* Get the WHERE clause if necessary */

     call get_where_clause(rel_index, where);

     /* Now create the SUM request from the data provided */

        /* Include the SUM clause in the request          */
        /* Include the selected attribute in the request */
        /* Include the FROM clause in the request         */
        /* Include the relation name in the request       */
        /* Include any WHERE clause in the request        */
end proc man_sum;

/*****************************************************************/

proc man_count(output: request);
/* This procedure creates a rrds COUNT request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                               */

     /* Get relation name for the request */

     t_write Generate COUNT request from which relation ?
     rel_index = get_rel();

     /* Get the WHERE clause if necessary */

     call get_where_clause(rel_index, where);

     /* Now create the COUNT request from the data provided */

        /* Include the COUNT clause in the request   */
        /* Include the FROM clause in the request    */
        /* Include the relation name in the request  */
        /* Include any WHERE clause in the request   */
end proc man_count;
/*******************************************************************/
```

```
proc man_delete(output: request);
/* This procedure creates a rrds DELETE  request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                                */

     /* Get relation name for the request */

     t_write Generate DELETE request from which relation ?
     rel_index = get_rel();
     t_write the two types of DELETE requests that can be created
     t_read type of DELETE request

     if type = 1 then
        /* Include DELETE ALL FROM clause in the request */
        /* Include the relation name in the request      */
     else if type = 2 then
        /* Get the WHERE clause if necessary */
        call get_where_clause(rel_index, where);
        /* Include DELETE FROM clause in the request */
        /* Include the relation name                 */
        /* Include the WHERE clause in the request   */
     end if
end proc man_delete;

/*******************************************************************/

proc man_select(output: request);
/* This procedure creates a rrds  SELECT request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                                */

     /* Get relation name for the request */

     t_write Generate SELECT request from which relation ?
     rel_index = get_rel();
     t_write the two types of SELECT requests that can be created
     t_read the type of SELECT request

     if type = 1 then
        /* Include the SELECT ALL FROM clause in the request */
        /* Include the relation name in the request          */
     else if type = 2 then
        /* Get the WHERE clause if necessary */
        call get_where_clause(rel_index, where);
        /* Include the SELECT FROM clause in the request */
        /* Include the relation name in the request      */
        /* Include any WHERE clause in the request       */
     end if
end proc man_select;
/*****************************************************************/
```

```
proc man_diff(output: request);
/* This procedure creates a rrds DIFF request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                             */

     /* Get relation name for the request */

     t_write Generate DIFF request from which relation ?
     rel1_index = get_rel();

     /* Get the second relation used in the request */

     t_write Generate DIFF request for which other relation ?
     rel2_index = get_rel();

     /* Now create the DIFF request from the data provided */

       /* Include the first relation name in the request  */
       /* Include the DIFF clause                         */
       /* Include the second relation name in the request */

end proc man_diff;

/******************************************************************/

proc man_intsect(output: request);
/* This procedure creates a rrds INTSECT request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                                */

     /* Get the first relation name for the request */

     t_write Generate INTSECT request from which relation ?
     rel1_index = get_rel();

     /* Get the second relation name for the request */

     t_write Generate INTSECT request for which other relation ?
     rel2_index = get_rel();

     /* Now create the INTSECT request from the data provided */

       /* Include the first relation name in the request  */
       /* Include the INTSECT clause in the request       */
       /* Include the second relation name in the request */

end proc man_intsect;

/******************************************************************/
```

```
proc man_join(output: request);
/* This procedure creates a rrds JOIN  request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                                */

     /* Get the first relation name for the request */

     t_write Generate JOIN request from which relation ?
     rel1_index = get_rel();

     /* Get the second relation name for the request */

     t_write Generate JOIN request for which other relation ?
     rel2_index = get_rel();

     /* Now create the JOIN request from the data provided */

       /* Include the first relation name in the request  */
       /* Include the JOIN clause in the request          */
       /* Include the second relation name in the request */

end proc man_join;

/*****************************************************************/

proc man_union(output: request);
/* This procedure creates a rrds UNION request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                                */

     /* Get the first relation name for the request */

     t_write Generate UNION request from which relation ?
     rel1_index = get_rel();

     /* Get the second relation name for the request */

     t_write Generate UNION request for which other relation ?
     rel2_index = get_rel();

     /* Now create the UNION request from the data provided */

       /* Include the first relation name in the request  */
       /* Include the UNION clause in the request          */
       /* Include the second relation name in the request */

end proc man_union;

/*****************************************************************/
```

```
proc man_insert(output: request);
/* This procedure creates a rrds INSERT request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                               */

     /* Get relation name for the request */

     t_write Generate INSERT request from which relation ?
     rel_index = get_rel();

     /* Get data for the attributes used in the request */

     t_write Enter data for each attribute.
     t_write NOTE: Character data must start with an upper case letter.
     for each attribute in the relation
         t_write the attribute name
         t_write the attribute type
         t_read the attribute value into an array
     end for

     /* Now create the INSERT request from the data provided */

          /* Include the INSERT clause in the request     */
          /* Include attribute value pairs in the request */
          /* Include the INTO clause in the request       */
          /* Include the relation name in the request     */

end proc man_insert;

/******************************************************************/
```

```
proc man_project(output: request);
/* This procedure creates a rrds PROJECT request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                                 */

     /* Get relation name for the request */

     t_write Generate PROJECT request from which relation ?
     rel_index = get_rel();

     /* Get the attributes used in the request */

     t_write Generate PROJECT request for which attributes ?
     for each attribute in the relation
         t_write the attribute name
         t_write a prompt for attribute inclusion
         t_read response into an array
     end for

     /* Get the WHERE clause if necessary */

     call get_where_clause(rel_index, where);

     /* Now create the PROJECT request from the data provided */

        /* Include the PROJECT clause in the request      */
        /* Include any selected attributes in the request */
        /* Include the FROM clause in the request         */
        /* Include the relation name in the request       */
        /* Include any WHERE clause in the request        */

end proc man_project;

/*******************************************************************/
```

```
proc man_update(output: request);
/* This procedure creates an rrds UPDATE request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                                */

     /* Get relation name for the request */

     t_write Generate UPDATE request from which relation ?
     rel_index = get_rel();

     /* Get data for the attributes used in the request */

     t_write Enter new data for each attribute or
                  hit 'return' for no change.
     t_write NOTE: Character data must start with an upper case letter.

     for each attribute in the relation
          t_write the attribute name
          t_write the attribute type
          t_read  the attribute value into an array
     end for

     /* Get the WHERE clause if applicable */

     call  get_where_clause(rel_index, where);

     /* Now create the UPDATE request from the data provided */

        /* Include the UPDATE clause in the request     */
        /* Include attribute value pairs in the request */
        /* Include the IN clause                        */
        /* Include the relation name in the request     */
        /* Include any WHERE clause in the request      */

end proc man_update;

/***********************************************************************/
```

```
proc edit_req_file();
/* This procedure allows a user to edit a specific rrds request file. */
/* A menu of editing options is displayed and executed.               */
/* GLOBAL:  req_cnt, req_buffer[]                                      */

     t_write Enter the name of a request file to be edited
     t_read the name of the request file
     f_read requests from the file into the request array
     while the user wants to edit the request file
         t_write menu of editing operations
         t_read operation selected from menu
         case editing operation
              DISPLAY REQUESTS:
                 call display_req();

              DELETE A REQUEST:
                 call delete_req();

              ADD A REQUEST:
                 call add_req();

              MODIFY A REQUEST:
                 call modify_req();

              QUIT:
                 /* exit the loop */

              OTHERWISE:
                 t_write an error message
         end case
     end while

     t_write SAVE all changes ?
     t_read user response

     if response = yes then
        f_write requests from request array into request file
     end if

end proc edit_req_file;

/*****************************************************************/
```

```
proc display_req();
/* This procedure allows a user to view all requests or selected  */
/* requests in a file.  Requests are displayed 5  at a time when  */
/* all requests must  be viewed.                                  */
/* GLOBAL:  req_cnt, req_buffer[]                                 */

     while the user wants to display requests
         t_write Enter a/s/q to display all, specific, or quit.
         t_read display mode
         if all requests are to be shown then
            t_write 5 requests at a time from the request buffer
         else if a specific request is desired then
            t_write Enter position
            t_read request position
            t_write request from position in request array
         end if
     end while

end proc display_req;

/*******************************************************************/

proc delete_req();
/* This procedure allows a user to delete requests  from a file. */
/* The user is prompted for the request to be deleted.           */
/* GLOBAL:  req_cnt, req_buffer[]                                */

     t_write each request in the request array
     while the user wants to delete requests
         t_write Enter position of request to be deleted.
         t_read position of request
         t_write request at selected position from request array
         t_write DELETE this request ?
         t_read  response
         if response = yes then
            /* Adjust array to overwrite the request */
         end if
         t_write Delete another request ?
     end while
end proc delete_req;

/************************************************************/
```

```
proc add_req();
/* This procedure allows the user to insert specific type of    */
/* rrds requests at a specific location in a request file.     */
/*  A menu of request types is displayed and created.          */
/* GLOBAL:  req_cnt, req_buffer[]                              */

     while the user wants to add requests and is below the limit
        t_write a menu of RRDS request types that can be added
        t_read the request type to be added
        t_write Enter file position for request insertion
        t_read position
        /* Make room in the request array for the new request. */
        case request type
             AVERAGE:
                call man_average(req_buffer[position].request);
                req_cnt = req_cnt + 1;
             COUNT:
                call man_count(req_buffer[position].request);
                req_cnt = req_cnt + 1;
             DELETE:
                call man_delete(req_buffer[position].request);
                req_cnt = req_cnt + 1;
             DIFF:
                call man_diff(req_buffer[position].request);
                req_cnt = req_cnt + 1;
             INSERT:
                call man_insert(req_buffer[position].request);
                req_cnt = req_cnt + 1;
             INTSECT:
                call man_intsect(req_buffer[position].request);
                req_cnt = req_cnt + 1;
             JOIN:
                call man_join(req_buffer[position].request);
                req_cnt = req_cnt + 1;
             MAX:
                call man_max(req_buffer[position].request);
                req_cnt = req_cnt + 1;;
             MIN:
                call man_min(req_buffer[position].request);
                req_cnt = req_cnt + 1;
             PROJECT:
                call man_project(req_buffer[position].request);
                req_cnt = req_cnt + 1;
             SELECT:
                call man_select(req_buffer[position].request);
                req_cnt = req_cnt + 1;
             SUM:
                call man_sum(req_buffer[position].request);
                req_cnt = req_cnt + 1;
```

```
            UNION:
               call man_union(req_buffer[position].request);
               req_cnt = req_cnt + 1;
            UPDATE:
               call man_update(req_buffer[position].request);
               req_cnt = req_cnt + 1;
            ADHOC:
               call man_adhoc(req_buffer[position].request);
               req_cnt = req_cnt + 1;
            QUIT:
               /* exit the loop */
            OTHERWISE:
               t_write error message
        end case
     end while
end proc add_req;

/************************************************************/

proc modify_req();
/* This procedure allows a user to modify  specific requests   */
/* from a file. The user is prompted for the character string  */
/* to be modified and the replacement string.                  */
/* GLOBAL:  req_cnt, req_buffer[]                              */

     while the user wants to modify requests
         t_write Enter position of request to be modified
         t_read position of request
         t_write the request from the position in the request array
         t_write Modify this request ?
         t_read response
         if response is yes then
            t_write Enter the string to be changed.
            t_read the old_string
            t_write Enter the new string
            t_read the new_string
            /* replace old string with the new string */
         end if
         t_write Modify another request ?
     end while
end proc modify_req;
/**************************************************************/
```

```
proc run_req_file();
/* This procedure allows a user to execute requests.          */
/* All the requests or selected requests can be passed to the */
/* Controller for execution. The results can be displayed on  */
/* a screen and stored into a specific output file.           */
/* GLOBAL:  req_cnt, req_buffer, output                        */

     t_write Enter name of request file to be used
                  for processing requests.
     t_read the file name
     f_read requests from the file into the request array
     t_write Enter s/f/b to direct output to a screen, file, or both.
     t_read the output location
     while the user wants to run requests
         t_write Enter a/s/q to run
                      all requests, selected requests, or quit
         t_read the mode
         if mode = a then
            call run_all();
         else if mode = s then
            call run_selected();
         end if
     end while
end proc run_req_file;

/********************************************************************/

proc run_all();
/* This procedure allows all rrds requests in the req_buffer to be */
/* processed by the Controller.                                     */
/* GLOBAL:  req_cnt, req_buffer[], output                           */

     for each request in the request array
         send the request to the controller for execution;
         receive results from the controller;
         if output = both or output = file then
            f_write  the request to the output file
            f_write  the results to the output file
         end if
         if output = both or output = screen then
            t_write the request
            t_write the results
         end if
     end for
end proc run_all;

/*******************************************************************/
```

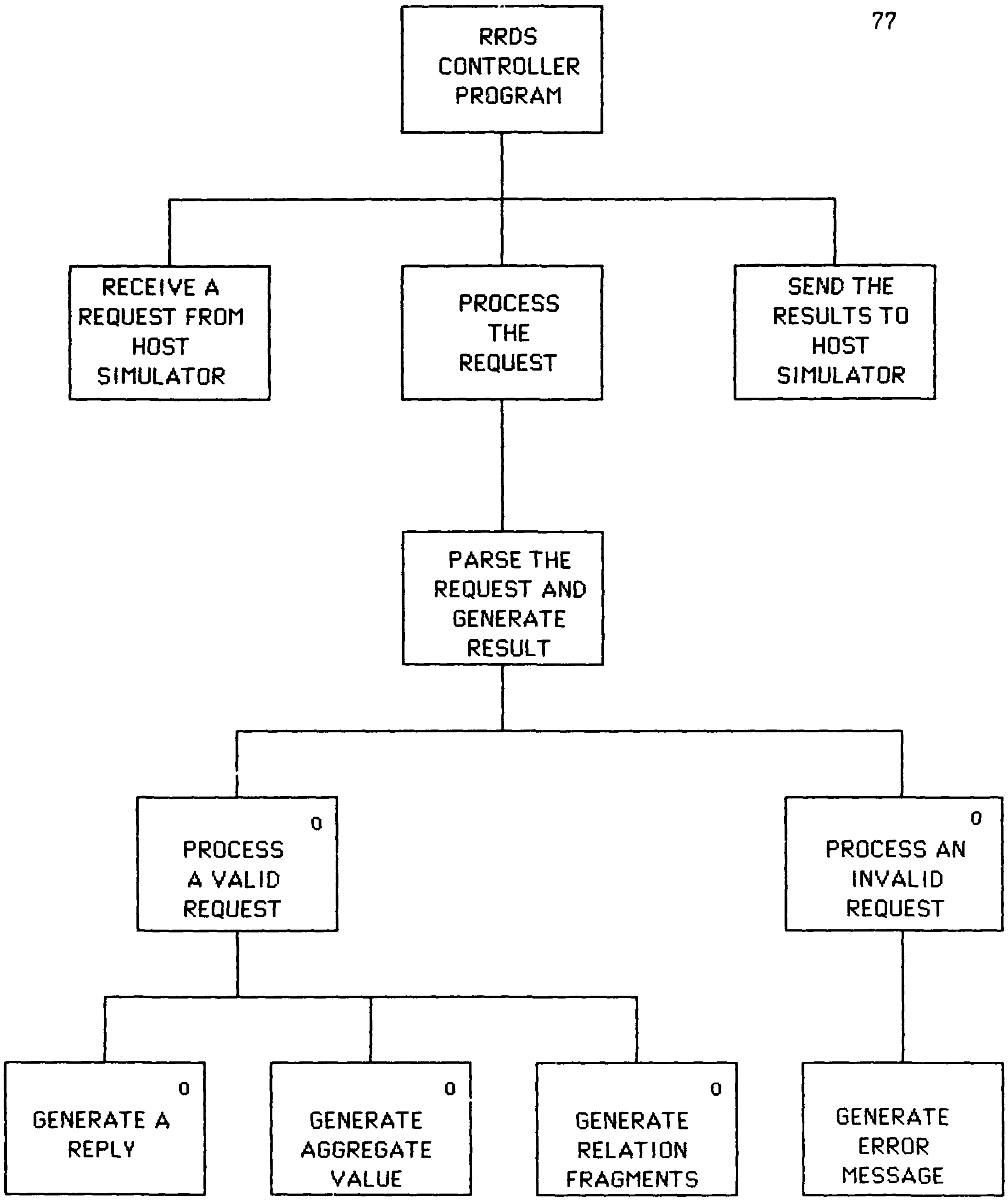
```
proc run_selected();
/* This procedure enables users to run  specific requests in the */
/* req_buffer or interactively create and run  requests.         */
/* GLOBAL:  req_cnt, req_buffer[], output                        */

     call display_req();
     t_write prompt for location of request (file or terminal)
     t_read location of request
     while the user wants to run selected requests
         if the request is in a file then
            t_write Enter position of request to be processed
            t_read position of request in the request array
            t_write the request
            t_write Run this request ?
            t_read response
            if response is yes then
               send the request to the controller for execution;
               receive results from the controller;
               if output = both or output = file then
                  f_write the request to the output file
                  f_write the results to the output file
               end if
               if output = both or output = screen then
                  t_write the request
                  t_write the results
               end if
            end if /* proper request */
         end if /* request in file */
```

```
if request is to be created at the terminal then
   t_write a menu of RRDS request types
   t_read selected request type
   case request type
         AVERAGE:
            call man_average(t_request);
         COUNT:
            call man_count(t_request);
         DELETE:
            call man_delete(t_request);
         DIFF:
            call man_diff(t_request);
         INSERT:
            call man_insert(t_request);
         INTSECT:
            call man_intsect(t_request);
         JOIN:
            call man_join(t_request);
         MAX:
            call man_max(t_request);
         MIN:
           call man_min(t_request);
         PROJECT:
            call man_project(t_request);
         SELECT:
            call man_select(t_request);
         SUM:
            call man_sum(t_request);
         UNION:
            call man_union(t_request);
         UPDATE:
            call man_update(t_request);
         ADHOC:
            call man_adhoc(t_request);
         QUIT:
            /* exit the case */
         OTHERWISE:
            t_write error message
   end case
```

```
            if request selection is valid then
               send the request to the controller for execution;
               receive results from the controller;
               if output = both or output = file then
                  f_write the request in t_request to the output file
                  f_write the results to the output file
               end if
               if output = both or output = screen then
                  t_write the request
                  t_write the results
               end if
            end if /* selection is valid */
         end if /* request is from terminal */
         t_write prompt to determine location of request
                       (file or terminal)
         t_read prompt
     end while /* more requests to run */
end proc run_selected;

/******************************************************/

func get_rel () returns (integer);
/* This function returns the index of a relation name */
/* in the rel_templates buffer                        */

     t_write list of relation names from templates
     t_read  rel_index of corresponding relation name
     return(rel_index);
end func get_rel;

/************************************************************/

proc get_where_clause (input: rel_index, output: clause);
/* This procedure generates the predicates and conjunctions */
/* for the specifier of a WHERE clause for a request        */

     t_write a WHERE clause prompt
     t_read user response
     while the user wants to include another conjunction in the specifier
           t_write prompt for predicates
           t_read user response
           while the user wants another predicate in this conjunction
                 t_write prompts for predicate data
                 t_read user responses
                 /* Include, in the clause,           */
                 /* the predicate for the conjunction */
           end while
           t_write prompt for another conjunction
           t_read user response
     end while
end proc get_where_clause;
```

APPENDIX D

JACKSON CHART AND PROGRAM DESIGN LANGUAGE FOR THE RRDS CONTROLLER PROGRAM


RRDS CONTROLLER PROGRAM
RECEIVE A REQUEST FROM HOST SIMULATOR
PROCESS THE REQUEST
SEND THE RESULTS TO HOST SIMULATOR
PARSE THE REQUEST AND GENERATE RESULT
O
PROCESS A VALID REQUEST
O
PROCESS AN INVALID REQUEST
O
GENERATE A REPLY
O
GENERATE AGGREGATE VALUE
O
GENERATE RELATION FRAGMENTS
GENERATE ERROR MESSAGE

```
/*****************************************************************/
/* This RRDS Controller program was developed and written       */
/* by Leonard A. Lyon during the Spring 1987 Semester.          */
/* This RRDS Controller is designed for use with a host         */
/* simulator.  This program will accept requests from the       */
/* host, parse the requests, generate RC result data,           */
/* coalesce the result data, and forward results to the host. */
/*****************************************************************/

proc main();
/* BEGIN RRDS CONTROLLER */

     /* Initialize string constants */

     TERMINATE = "@";
     CONTINUE  = "+";

     /* Establish a communication path with the host simulator */

     /* Wait to receive  the first request from the host  */
     /* Process  requests until the host terminates */

     while the host has more requests
           t_write the request
           yyin = input request;     /* Set lex input ptr */
           valid_req = yyparse();    /* Invoke the yacc parser */
           if valid_req = Ø then
              /* Process a valid request */
              /* Simulate broadcasting a request by */
              /*  writing to BROADCAST file */

              f_write the request to the BROADCAST file

              /* Simulate receiving results from RCs */

              call get_results();
           else  /* an invalid request */
              t_write an error message

              /* Now generate error message for host */

              call bad_req();
           end if

           /* Transmit the results of the request to the host */

           call send_results();

           /* wait to receive the next request from the host  */

     end while
end proc main;
```

```
proc get_results();
/* This procedure examines each request to */
/* determine the type of RC result data to simulate */

     if recv_request = "DELETE" then
        reply = "VALID DELETE COMPLETE";
     else
     if recv_request = "INSERT" then
        reply = "VALID INSERT COMPLETE";
     else
     if recv_request = "UPDATE" then
        reply = "VALID UPDATE COMPLETE";
     else
     if recv_request = "AVERAGE" then
        call co_AVG();
     else
     if recv_request = "COUNT" then
        call co_COUNT();
     else
     if recv_request =  "MAX" then
        call co_MAX();
     else
     if recv_request = "MIN" then
        call co_MIN();
     else
     if recv_request = "SUM" then
        call co_SUM();
     else
     if recv_request = "PROJECT" then
        call co_RECORDS();
     else
     if recv_request = "SELECT" then
        call co_RECORDS();
     else
     if recv_request = "DIFF" then
        call co_RECORDS();
     else
     if recv_request = "INTSECT" then
        call co_RECORDS();
     else
     if recv_request = "JOIN" then
        call co_RECORDS();
     else
     if recv_request = "UNION" then
        call co_RECORDS();
     end if
end proc get_results;
```

```
proc co_AVG();
/* Coalesce the sums and counts from */
/* the RCs to create the average value*/

     /* Get simulated RC data from files */

     for each RC result data file
         f_read a count value into count_datum from result file
         f_read a sum value into sum_datum from result file
         count = count + count_datum;
         sum = sum + sum_datum;
     end for

     /* Compute the true average */

     avg = sum / count;

     /* Generate the reply message */

     reply = avg;
end proc co_AVG;

/*****************************************************/

proc co_COUNT();
/* Coalesce the count from the RCs */

     /* Get simulated RC data from files */

     for each RC result data file
         f_read the count from result file  into datum
         /* Compute the total count */
         count = count + datum;
     end for

     /* Generate the reply message */

     reply = count;
end proc co_COUNT;
```

```
proc co_MAX();
/* Coalesce the max values from the RCs */

     /* Get simulated RC data from files */

     f_read the value from the first RC file into max_val

     if max_val = string data then
        /* Process string data */

        for each remaining RC file
            /* Coalesce the max string values */
            f_read the value
            if value > max_val then
               max_val = value
            end if
        end for

        /* Generate the reply message */

        reply = max_val;
     else
        /* Process numeric data */
        /* max_val is a numeric stored as string */

        max_number = numeric value of max_val;
        for each remaining RC data file
            /* Coalesce the max digit values */
            f_read the value int datum
            number = numeric value of datum;
            if number > max_number then
               max_number = number;
            end if
        end for

        /* Generate the reply message */

        reply = max_number
     end if
end proc co_MAX;
```

```
proc co_MIN();
/* Coalesce the min values from the RCs */

     /* Get simulated RC data from files */


     f_read the value from the first RC result file into min_val
     if min_val is string data then
        /* Process string data */

        for each remaining RC result file
            /* Coalesce the min string values */
            f_read the value from the result file
            if value < min_val then
               min_val = value;
            end if
        end for

        /* Generate the reply message */
        reply = min_val;
     else

        /* Process numeric data */
        /* min_val is a numeric stored as string */

        min_number = numeric value of min_val;
        for each remaining RC file
            /* Coalesce the min digit values */
            f_read the value into datum
            number = numeric value of datum;
            if number < min_number then
               min_number = number;
            end if
        end for

       /* Generate the reply message */

       reply = min_number;
    end if
end proc co_MIN;
```

```
proc co_SUM();
/* Coalesce the sum values from the RCs */

     /* Get simulated RC data from files */

     for each RC result file
         /* Compute the total sum */
         f_read a value from a result file into datum
         sum = sum + datum;
     end for

     /* Generate the reply message */

     reply = sum;
end proc co_SUM;
```

```
proc co_RECORDS();
/* Coalesce the relation fragments from the RCs */

     /* SIMULATE retrieving data from each RC via RC data files */

     for each RC result file
         f_read a random number of records in the  file
                into a result buffer (RC_buffer)
     end for

     /* Now coalesce the relation fragment records      */
     /* from all of the RC's. Take each record and      */
     /* remove its duplicates in the rest of the buffer */

     buf_limit = number of records in the buffer;
     compare_index = 0;
     while compare_index < buf_limit
           compare_buffer = the record in RC_buffer[compare_index];

           /* Go through the rest of the buffer and remove duplicates */

           buf_index = compare_index + 1;
           while buf_index < buf_limit
             if compare_buffer = the record in RC_buffer[buf_index] then
                /* A duplicate relation fragment exists   */
                /* Remove the duplicate by overwriting it */
                /* There is now one less record in the buffer */
                buf_limit = buf_limit - 1;
             end if
             buf_index = buf_index + 1;
           end while
           compare_index =  compare_index + 1;
     end while

     /* Generate a reply message */

     if buf_limit > 0 then
        reply = "There are some results";
     else
        reply = "VALID REQUEST -- No Result";
     end if
end proc co_RECORDS;
```

```
proc bad_req();
/* Generates an error message */

     reply = "INVALID REQUEST ";
end proc bad_req;


/**********************************************************************/

proc send_results();
/* This procedure transmits the results of a request to the host    */
/* over the communication path                                       */

     if reply = "There are some results" then
        /* Send coalesced relation fragments to the host from RC_buffer*/
        send the "reply" to the host;
        for each record in the result buffer
            send the record to the host;
        end for
        send an end of result message to the host;
     else
        /* send back only one result */
        send the "reply" to the host;
     end if
end proc send_results;
```

APPENDIX E

USER / MAINTENANCE GUIDE

E.1. User Guide

The host simulator program provides the capability to generate, edit, and run RRDS requests. The host simulator program is invoked by entering "host.out". Then, a menu of operations is displayed to allow a user to:

(1) Create a relation template.

(2) Display existing relation templates.

(3) Create a file of RRDS requests.

(4) Display the requests in a request file.

(5) Edit a request file.

(6) Send requests to the controller program for execution.

The program is menu driven and prompts the user for any necessary information. Thus, no special commands must be known to use the host simulator program. However, at least one relation template must exist before requests can be generated by the program.

When the option to process requests is selected from the menu, the controller program must be invoked by entering "rrds.out" at a separate terminal. The controller program will accept requests from the host, one at a time, parse the request, generate request results, and forward results to the host simulator program. For proper operation, the executable files, "host.out" and "rrds.out", must be located in the same directory to establish the socket channel, "busl", used for interprocess communication between the two programs.

In addition to RRDS request files and output files, other files are created and used during the operation of the host simulator and controller programs. This includes the "TEMPLATES" file, created by the host to store relation templates created by the user, the "BROADCAST" file, created by the controller to simulate transmitting valid requests to the RCs, and the result data files ("RC_0.data", "RC_1.data", "RC_2.data", and "RC_3.data"), used by the controller to generate and simulate request results of from the RCs.

If the request files, result files, and "TEMPLATES" file are damaged as the result of a system crash or quota overflow error, they must be recreated. If the host simulator program is aborted, the file "busl" may need to be removed before rerunning the host simulator program. This can be easily done by entering "rm busl".

E.2. Maintenance Guide

Any of the files shown in Table 2 can be modified to alter the operations of the host simulator or controller program.

The lexical analyzer generated by LEX, "lex.yy.c", reads input from the terminal. The analyzer must be changed to read

File name	Description
host.c	source code for the host simulator program
rrds.c	source code for the RRDS controller program
rrds.vars	common constants and variables for both programs
rrds.functions	common functions used by both programs
parse	YACC specification code for the parser
tokens	LEX specification code for the lexical analyzer
lex.yy.c	LEX output source code (modified for RRDS)
y.tab.c	YACC output source code

Table 2: The Files for the Host Simulator Program and the RRDS Controller Program

input from the "recv_request" buffer (this buffer is set in the controller program, "rrds.c", and contains the request from the host). The lexical analyzer, "lex.yy.c", can be changed to read request characters from the buffer as follows:

(1) Comment out the "FILE" declaration of "yyin". ("yyin" is redefined in "rrds.c".)

(2) Append "++" increment symbols to "yyin" in the declaration of the "input" function.

(3) Change "getc" to "get_rrds_req_ch" in the declaration of the "input" function so that input is not taken from the terminal.

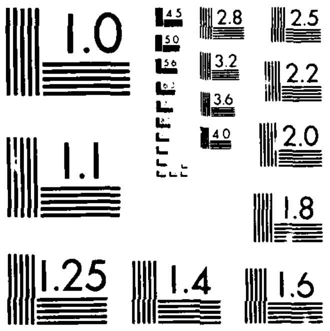
(4) Define "get_rrds_req_ch" function as follows:

```
get_rrds_req_ch(p)
char *p;
{
 if(*p=='\0') return(EOF);
 return(0377 & *p);
}
```

The "get_rrds_req_ch" function returns a bit mapped character from the current character pointer. (This pointer, "p", is set in "rrds.c" to the beginning of the "recv_request" buffer and is incremented when the lexical analyzer invokes the "input" function.)

AD-A190 951 DESIGN AND IMPLEMENTATION OF A CONTROLLER AND A HOST SIMULATOR FOR A RELA..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH L A LYON 1987 AFIT/CI/NR-88-61 2/2

UNCLASSIFIED F/G 12/5 NL


1.0
4.5
5.0
5.6
2.8
2.5
3.2
2.2
3.6
4.0
2.0
1.1
1.8
1.25
1.4
1.6

APPENDIX F

LEX SPECIFICATION CODE

This appendix presents the tokens that are used in the LEX specification code to create the lexical analyzer, "yylex()". These tokens are derived RRDS requests and are passed to the parser, "yyparse()".

```
%%
"ALL"         { return(ALL); }
"AND"         { return(AND); }
"FROM"        { return(FROM); }
"INTO"        { return(INTO); }
"IN"          { return(IN); }
"OR"          { return(OR); }
"WHERE"       { return(WHERE); }
"AVERAGE"     { return(AVERAGE); }
"COUNT"       { return(COUNT); }
"MAX"         { return(MAX); }
"MIN"         { return(MIN); }
"SUM"         { return(SUM); }
"DIFF"        { return(DIFF); }
"DELETE"      { return(DELETE); }
"INSERT"      { return(INSERT); }
"INTSECT"     { return(INTSECT); }
"JOIN"        { return(JOIN); }
"PROJECT"     { return(PROJECT); }
"SELECT"      { return(SELECT); }
"UNION"       { return(UNION); }
"UPDATE"      { return(UPDATE); }
```

```
[A-Z]           { return(UC_LETTER); }
[a-z]           { return(LC_LETTER); }
[0-9]           { return(DIGIT); }
":="            { return(ASSIGN_OP); }
"+"             |
"-"             { return(ADD_OP); }
"*"             |
"/"             { return(MUL_OP); }
"<="            |
">="            |
"<>"            |
"<"             |
">"             |
"="             { return(REL_OP); }
"."             { return(PERIOD); }
","             { return(COMMA); }
"["             { return(LBRACKET); }
"]"             { return(RBRACKET); }
"("             { return(LPAREN); }
")"             { return(RPAREN); }
[ \t\f\n]       ;
.               { return(BAD); }
%%
```

APPENDIX G

YACC SPECIFICATION CODE

This appendix contains the "YACC" specification code that is used to generate the parser for the controller program, "yyparse()". The parser accepts the tokens from the lexical analyzer, "yylex()", and matches them with the "YACC" syntax specification code shown below. This syntax specification code is based on the BNF description of the RRDS data manipulation language presented in Appendix A.

```
%token ALL AND ASSIGN_OP BAD
%token FROM IN INTO OR WHERE
%token AVERAGE COUNT MAX MIN SUM
%token DIFF DELETE JOIN
%token INSERT INTSECT PROJECT
%token SELECT UNION UPDATE
%token DIGIT LC_LETTER UC_LETTER
%token PERIOD COMMA LBRACKET RBRACKET LPAREN RPAREN
%left  ADD_OP MUL_OP REL_OP

%%
Request         : LBRACKET Query RBRACKET
                ;
Query           : Select_Query   | Project_Query
                | Insert_Query   | Delete_Query
                | Update_Query   | Join_Query
                | Union_Query    | Diff_Query
                | Intsect_Query  | Agg_Query
                ;
Agg_Query       : Ave_Agg | Count_Agg | Max_Agg
                | Min_Agg | Sum_Agg
                ;
Select_Query    : SELECT FROM Target_Relation
                         WHERE LPAREN Specifier RPAREN
                | SELECT ALL FROM Target_Relation
                ;
Project_Query   : PROJECT LPAREN Attr_List RPAREN FROM Target_Relation
                | PROJECT LPAREN Attr_List RPAREN FROM Target_Relation
                          WHERE LPAREN Specifier RPAREN
                ;
```

```
Insert_Query   : INSERT LPAREN Record RPAREN INTO Target_Relation
               ;
Delete_Query   : DELETE FROM Target_Relation
                        WHERE LPAREN Specifier RPAREN
               | DELETE ALL FROM Target_Relation
               ;
Update_Query   : UPDATE LPAREN Modifier RPAREN IN Target_Relation
               | UPDATE LPAREN Modifier RPAREN IN Target_Relation
                        WHERE LPAREN Specifier RPAREN
               ;
Diff_Query     : Target_Relation DIFF Target_Relation
               ;
Intsect_Query  : Target_Relation INTSECT Target_Relation
               ;
Join_Query     : Target_Relation JOIN Target_Relation
               ;
Union_Query    : Target_Relation UNION Target_Relation
               ;
Ave_Agg        : AVERAGE LPAREN Attribute RPAREN FROM Target_Relation
               | AVERAGE LPAREN Attribute RPAREN FROM Target_Relation
                         WHERE LPAREN Specifier RPAREN
               ;
Count_Agg      : COUNT FROM Target_Relation
               | COUNT FROM Target_Relation
                       WHERE LPAREN Specifier RPAREN
               ;
Max_Agg        : MAX LPAREN Attribute RPAREN FROM Target_Relation
               | MAX LPAREN Attribute RPAREN FROM Target_Relation
                     WHERE LPAREN Specifier RPAREN
               ;
Min_Agg        : MIN LPAREN Attribute RPAREN FROM Target_Relation
               | MIN LPAREN Attribute RPAREN FROM Target_Relation
                     WHERE LPAREN Specifier RPAREN
               ;
Sum_Agg        : SUM LPAREN Attribute RPAREN FROM Target_Relation
               | SUM LPAREN Attribute RPAREN FROM Target_Relation
                     WHERE LPAREN Specifier RPAREN
               ;
Target_Relation : String
                ;
Record          : Record_Segment
                ;
Record_Segment  : Attr_Value_Pair
                | Record_Segment COMMA Attr_Value_Pair
                ;
Attr_Value_Pair : Attribute ASSIGN_OP Value
                ;
Attr_List      : Attribute
               | Attr_List COMMA Attribute
               ;
Attribute      : String
               ;
```

```
Specifier       : Conjunction
                | Specifier OR Conjunction
                ;
Conjunction     : LPAREN Pred_List RPAREN
                ;
Pred_List       : Predicate
                | Pred_List AND Predicate
                ;
Predicate       : Attribute REL_OP Value
                ;
Modifier        : Assign_Statement_List
                ;
Assign_Statement_List : Assign_Statement
                      | Assign_Statement_List COMMA Assign_Statement
                      ;
Assign_Statement      : Attribute ASSIGN_OP Arithm_Expression
                      ;
Arithm_Expression     : Term
                      | Arithm_Expression ADD_OP Term
                      ;
Term            : Factor
                | Term MUL_OP Factor
                ;
/* Factor       : Attribute | Value                   */
/*              | LPAREN Arithm_Expression RPAREN     */
/*              ;                                     */
/*****************************************************/
/*                                                   */
/* The actual BNF for Factor is shown above.         */
/* However, "Attribute" was eliminated from the      */
/* YACC implementation to eliminate the              */
/* "reduce/reduce" ambiguity conflicts, as           */
/* shown below.  The conflict occurs because         */
/* "Attribute" and "Value" are both defined as       */
/* a "String". YACC's default conflict resolution */
/* would cause "Attribute" to be chosen for          */
/* "Attribute" or "Value".                           */
/*                                                   */
/*****************************************************/
Factor          : Value | LPAREN Arithm_Expression RPAREN
                ;
```

```
Value           : String | Number
                ;
String          : UC_LETTER | UC_LETTER Alph_Num
                ;
Alph_Num        : Letter | DIGIT
                | Alph_Num Letter
                | Alph_Num DIGIT
                ;
Number          : Integer | Real | ADD_OP Integer | ADD_OP Real
                ;
Real            : Integer PERIOD Integer
                ;
Integer         : DIGIT
                | Integer DIGIT
                ;
Letter          : UC_LETTER | LC_LETTER
                ;
%%
/*                                */
/* Include the lexical analyzer */
#include "lex.yy.c"
```

APPENDIX H

C LANGUAGE SOURCE CODE FOR THE HOST SIMULATOR PROGRAM

```
/**********************************************************/
/*                                                        */
/*                    HOST SIMULATOR                      */
/*                                                        */
/**********************************************************/
/*                                                        */
/*               Developed and Written by                 */
/*                   LEONARD A. LYON                      */
/*                     Spring 1987                        */
/*                                                        */
/*  This host simulator  program is designed for use      */
/*  with the RRDS Controller.  This program is menu       */
/*  driven and allows users to interactively create,      */
/*  edit, and run RRDS requests. Requests  are passed     */
/*  to the  Controller one at a time via a socket         */
/*  channel.  Request results are received from the       */
/*  Controller and can be logged in a file,               */
/*  displayed at the terminal, or both.                   */
/*                                                        */
/**********************************************************/

#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include "rrds.vars"
#include "rrds.functions"
```

```
/* BEGIN HOST SIMULATOR */

main()
{
 int rel_index;
 int att_index;
 int select;
 char *getlogin();

/* Initialize the request delimters and other constants */

 strcpy(BLANK, " ");
 strcpy(BOR, "[");
 strcpy(CONTINUE, "+");
 strcpy(EOR, "]");
 strcpy(TERMINATE, "@");
 strcpy(REL[0].OP, "<=");
 strcpy(REL[1].OP, ">=");
 strcpy(REL[2].OP, "<>");
 strcpy(REL[3].OP, ">");
 strcpy(REL[4].OP, "<");
 strcpy(REL[5].OP, "=");

/* Retrieve relation templates from TEMPLATE file */

 tmpl_cnt     = 0;
 strcpy(tmpl_name, "TEMPLATES");

 tmpl_desc = fopen(tmpl_name, "r");
 if(tmpl_desc != NULL)
    {
     while(fscanf(tmpl_desc, "%s %d",
                  &rel_templates[tmpl_cnt].rel_name[0],
                  &rel_templates[tmpl_cnt].rel_nof_attrs) != EOF)
                 {
                  att_index = 0;
                  while(att_index < rel_templates[tmpl_cnt].rel_nof_attrs)
                     {
                      fscanf(tmpl_desc, "%s %c",
                        &rel_templates[tmpl_cnt].rel_attrs[att_index].at_name[0],
                        &rel_templates[tmpl_cnt].rel_attrs[att_index].at_type);
                      ++ att_index;
                     }
                  ++ tmpl_cnt;
                 }
  fclose(tmpl_desc);
 }
```

```
active_bus = 0;
select = 1;
while (select != QUIT)
    {
     printf("**************************************************\n");
     printf("******                                      ******\n");
     printf("******           HELLO %8s              ******\n",getlogin());
     printf("******                                      ******\n");
     printf("******           WELCOME TO THE             ******\n");
     printf("******                                      ******\n");
     printf("******       HOST SIMULATOR (Level 1)       ******\n");
     printf("******                                      ******\n");
     printf("**************************************************\n\n");
     printf("Please select a number for the desired operation.\n");
     printf("[1]  CREATE  a relation template \n");
     printf("[2]  DISPLAY relation templates \n");
     printf("[3]  CREATE  a request file \n");
     printf("[4]  DISPLAY a request file \n");
     printf("[5]  EDIT    a request file \n");
     printf("[6]  RUN     a request file \n");
     printf("[%d]  QUIT \n", QUIT);
     prompt();
     scanf("%d", &select);
     getchar();                    /* read '\n' */
     switch(select)
           {
            case 1: create_template();
                    break;
            case 2: show_templates();
                    break;
            case 3: create_req_file();
                    break;
            case 4: show_req_file();
                    break;
            case 5: edit_req_file();
                    break;
            case 6: run_req_file();
                    break;
            case QUIT: break;
            default: printf("INVALID SELECTION  %d \n", select);
           }
    }
```

```
/* Check if channel  "bus1" was created and connected to Controller */

 if (active_bus == 1)
    disconnect_bus();

/* Save relation template descriptions */

 if (tmpl_cnt > 0)
    {
     tmpl_desc = fopen(tmpl_name, "w");
     rel_index = 0;
     while (rel_index < tmpl_cnt)
           {
            fprintf(tmpl_desc, "%s %d\n",
                    rel_templates[rel_index].rel_name,
                    rel_templates[rel_index].rel_nof_attrs);
            att_index = 0;
            while(att_index < rel_templates[rel_index].rel_nof_attrs)
                 {
                  fprintf(tmpl_desc, "%s %c\n",
                          rel_templates[rel_index].rel_attrs[att_index].at_name,
                          rel_templates[rel_index].rel_attrs[att_index].at_type);
                   ++ att_index;
                  }
            ++ rel_index;
           }
      fclose(tmpl_desc);
    }
}
/*****************************************************************************/
```

```
create_template()
/* This procedure displays existing relation templates and allows */
/* the user to create new relation templates.                     */
/* GLOBAL:  tmpl_cnt, rel_templates[]                             */
{
 int at_pos;            /* attribute index */
 int rel_index;         /* relation index  */
 char select[2];        /* user's response */
 char name[NAME_SIZE]; /* input buffer for relation & attribute names */


/* Display existing relation templates */

 rel_index = 0;
 while(rel_index < tmpl_cnt)
       {
        printf("RELATION:  %s \n",rel_templates[rel_index].rel_name);
        at_pos = 0;
        while(at_pos < rel_templates[rel_index].rel_nof_attrs)
            {
             printf("           ");
             printf("%s", rel_templates[rel_index].rel_attrs[at_pos].at_name);
             printf(" (%c)", rel_templates[rel_index].rel_attrs[at_pos].at_type);
             printf("\n");
             ++ at_pos;
            }
        printf("More ?  (y/n) \n");
        prompt();
        gets(select);
        if ((select[0] == 'n') || (select[0] == 'N') ||
            (select[0] == 'q') || (select[0] == 'Q'))
             break;
        ++ rel_index;
       }
/* Create the name of the new relation template */

 printf("Create another relation template ?  (y/n) \n");
 prompt();
 gets(select);
 while((select[Ø] == 'y' || select[Ø] == 'Y') && tmpl_cnt < MAX_RELS)
      {
       printf("Enter name of relation template. \n");
       prompt();
       gets(name);
       strcpy(rel_templates[tmpl_cnt].rel_name, name);

/* Create the attribute names for the new relation template */

       at_pos = Ø;
       while(at_pos < MAX_ATTRS)
            {
             printf("Enter name of attribute or hit 'RETURN' key\n");
             prompt();
             gets(name);
             if(name[Ø] == '\Ø')
                /* user is finished */
                break;
             else
                {
                /* Process valid name */

                strcpy(rel_templates[tmpl_cnt].rel_attrs[at_pos].at_name,name);

                printf("Enter attribute type (C/I) \n");
                prompt();
                gets(name);
                rel_templates[tmpl_cnt].rel_attrs[at_pos].at_type = name[Ø];
                ++ at_pos;
                }
            }
       rel_templates[tmpl_cnt].rel_nof_attrs = at_pos;
       ++ tmpl_cnt;
       printf("Create another relation template ? \n");
       prompt();
       gets(select);
      }
 }
/*********************************************************************************/
```

```
create_req_file()
/* This procedure allows the user to store rrds requests that are created */
/* either manually or automatically by the system.                        */
/* GLOBAL: req_cnt, req_buffer[]                                          */
{
int index;
static char selection[2];
req_cnt = 0;

printf("Enter the file name to be created \n");
prompt();
gets(file_name);

if (file_name[0] == '\0')
   return;

file_desc = fopen(file_name, "w");

if (file_desc == NULL)
   {
    printf("ERROR: Cannot create %s \n", file_name);
    return;
   }

printf("Enter 'a' or 'm' for automatic or manual request generation \n");
prompt();
gets(selection);

if (selection[0] == 'a' || selection[0] == 'A')
    auto_req_gen();
if (selection[0] == 'm' || selection[0] == 'M')
    man_req_gen();

/* Save all created requests */

index = 0;
while (index < req_cnt)
      {
       fprintf(file_desc, "%s\n", req_buffer[index].request);
       ++ index;
      }

fclose(file_desc);
}

/****************************************************************************/
```

```
auto_req_gen()
/* This procedure provides a menu of requests to be created automatically */
/* by the system and runs the selected request generator program.         */
{
int selection;
selection = 1;
while (selection != QUIT)
      {
       printf("Select type of request to be generated:      \n");
       printf("[1] INSERT   \n");
       printf("[%d] QUIT     \n",QUIT);
       prompt();
       scanf("%d", &selection);
       getchar();     /* Read \n */
       switch(selection)
             {
              case 1: auto_insert();
                      break;
              case QUIT: break;
              default: printf("INVALID SELECTION %d \n", selection);
             }
      }
}

/*****************************************************************************/
```

```
auto_insert()
/* This procedure creates a set of rrds INSERT requests with user input */
/* GLOBAL: tmpl_cnt, rel_templates[], req_cnt, req_buffer[]             */

{
 int at_index;                 /* attribute index in relation template    */
 int auto_index;               /* index for requested data to create recs */
 int choice;                   /* random index value for data selection   */
 int gen_index;                /* index for the number of requests genned */
 int rel_index;                /* relation index in relation template     */
 int req_index;                /* index to request array in req_buffer    */
 int req_bound;                /* number of requests to be created        */
 char value[VAL_SIZE];         /* derived from lower & upper bounds       */
 int input_bound[MAX_ATTRS];   /* number of string data values            */

 struct {
         char data[VAL_SIZE];
        } str_input[MAX_ATTRS] [MAX_AUTO];

 struct {
         char lower[1Ø];
         char upper[1Ø];
        } num_input[MAX_ATTRS];

 /* Get relation name for the request */

 printf("Generate INSERT requests from which relation ?\n");
 rel_index = get_rel();

 /* Determine how many requests to create */

 req_bound = -1;
 while(req_bound <= Ø || req_bound > MAX_GEN)
      {
       printf("How many requests do you want to create ? ( <= %d)\n", MAX_GEN);
       scanf("%d", &req_bound);
       getchar(); /* Read \n */
      }
 /* Determine total string data values the user will provide per attribute */

 for(at_index = Ø;at_index < rel_templates[rel_index].rel_nof_attrs;++at_index)
     input_bound[at_index] = -1;
```

```
for(at_index = 0;at_index < rel_templates[rel_index].rel_nof_attrs;++at_index)
   {
    while(input_bound[at_index] < 0 || input_bound[at_index] > MAX_AUTO)
         {
          printf("How many data values do you want to provide for %s ?",
                  rel_templates[rel_index].rel_attrs[at_index].at_name);

          if(rel_templates[rel_index].rel_attrs[at_index].at_type == 'i' ||
             rel_templates[rel_index].rel_attrs[at_index].at_type == 'I')
                printf(" 0 for range OR ");

          printf(" ( <= %d)\n", MAX_AUTO);
          scanf("%d", &input_bound[at_index]);
          getchar(); /* Read \n */
         }
   }

/* Get data for the attributes used in the request */

at_index = 0;
while(at_index < rel_templates[rel_index].rel_nof_attrs)
     {
      if(input_bound[at_index] != 0)
        {
         printf("Enter data at each prompt to create records.\n");
         printf("NOTE: Character data must start with an upper case letter.");
         printf("\n");
         auto_index = 0;
         while(auto_index < input_bound[at_index])
              {
               printf("%20s   ",
                      rel_templates[rel_index].rel_attrs[at_index].at_name);
               printf("(%c) ",
                      rel_templates[rel_index].rel_attrs[at_index].at_type);
               prompt();
               gets(str_input[at_index][auto_index].data);
               ++ auto_index;
              }
        }
      if((rel_templates[rel_index].rel_attrs[at_index].at_type == 'I' ||
          rel_templates[rel_index].rel_attrs[at_index].at_type == 'i') &&
          (input_bound[at_index] == 0))
        {
         printf("Enter a numeric lower bound for %s. ",
                rel_templates[rel_index].rel_attrs[at_index].at_name);
         prompt();
         gets(num_input[at_index].lower);
         printf("Enter a numeric upper bound for %s. ",
                rel_templates[rel_index].rel_attrs[at_index].at_name);
         prompt();
         gets(num_input[at_index].upper);
        }
      ++ at_index;
     }
printf("\n\n");
```

```
/* Now create the insert requests from the data provided */

gen_index = 0;
while(gen_index < req_bound && req_cnt < MAX_REQ)
     {
      strcpy(req_buffer[req_cnt].request, BOR);
      strcat(req_buffer[req_cnt].request, "INSERT (");
      at_index  = 0;

      /* Include attribute value pairs in the request */

      while(at_index < rel_templates[rel_index].rel_nof_attrs)
           {
            if(at_index > 0)
               strcat(req_buffer[req_cnt].request, ",");
            strcat(req_buffer[req_cnt].request,
                     rel_templates[rel_index].rel_attrs[at_index].at_name);
            strcat(req_buffer[req_cnt].request, " := ");
            if(input_bound[at_index] != 0)
              {
               choice = random(input_bound[at_index]);
               strcat(req_buffer[req_cnt].request,
                        str_input[at_index][choice].data);
              }
            if((rel_templates[rel_index].rel_attrs[at_index].at_type== 'I' ||
                rel_templates[rel_index].rel_attrs[at_index].at_type== 'i') &&
                  (input_bound[at_index] == 0))
              {
               gendigits(&value[0], &num_input[at_index].lower[0],
                                    &num_input[at_index].upper[0]);
               strcat(req_buffer[req_cnt].request, value);
              }
            ++ at_index;
           }

      /* Include the INTO clause in the request */

      strcat(req_buffer[req_cnt].request, ") INTO ");

      /* Include the relation name in the request */

      strcat(req_buffer[req_cnt].request, rel_templates[rel_index].rel_name);

      /* Add the EOR to the request */

      strcat(req_buffer[req_cnt].request, EOR);
      printf("%s\n", req_buffer[req_cnt].request);
      ++ req_cnt;
      ++ gen_index;
     }
printf("\n\n");
}

/**************************************************************************/
```

```
man_req_gen()
/* This procedure provides a menu of rrds request types that are   */
/* created by the user.  The appropriate request generator program */
/* is executed based on user input.                                */
/* GLOBAL: req_cnt, req_buffer[]                                   */
{
int select;

select = 1;
while (select != QUIT && req_cnt < MAX_REQ)
      {
       printf("Please select a number to create the desired request.\n");
       printf("[1] AVERAGE     [5] INSERT    [9]  MIN      [13] UNION  \n");
       printf("[2] COUNT       [6] INTSECT   [10] PROJECT  [14] UPDATE \n");
       printf("[3] DELETE      [7] JOIN      [11] SELECT   [15] ADHOC/ERROR\n");
       printf("[4] DIFF        [8] MAX       [12] SUM      [%d] QUIT \n",QUIT);

       prompt();
       scanf("%d", &select);
       getchar();   /* Read \n */
       switch (select)
              {
              case 1:  man_average(req_buffer[req_cnt].request);
                       ++ req_cnt;
                       break;
              case 2:  man_count(req_buffer[req_cnt].request);
                       ++ req_cnt;
                       break;
              case 3:  man_delete(req_buffer[req_cnt].request);
                       ++ req_cnt;
                       break;
              case 4:  man_diff(req_buffer[req_cnt].request);
                       ++ req_cnt;
                       break;
              case 5:  man_insert(req_buffer[req_cnt].request);
                       ++ req_cnt;
                       break;
              case 6:  man_intsect(req_buffer[req_cnt].request);
                       ++ req_cnt;
                       break;
              case 7:  man_join(req_buffer[req_cnt].request);
                       ++ req_cnt;
                       break;
              case 8:  man_max(req_buffer[req_cnt].request);
                       ++ req_cnt;
                       break;
              case 9:  man_min(req_buffer[req_cnt].request);
                       ++ req_cnt;
                       break;
              case 10: man_project(req_buffer[req_cnt].request);
                       ++ req_cnt;
                       break;
```

```
                case 11: man_select(req_buffer[req_cnt].request);
                         ++ req_cnt;
                         break;
                case 12: man_sum(req_buffer[req_cnt].request);
                         ++ req_cnt;
                         break;
                case 13: man_union(req_buffer[req_cnt].request);
                         ++ req_cnt;
                         break;
                case 14: man_update(req_buffer[req_cnt].request);
                         ++ req_cnt;
                         break;
                case 15: man_adhoc(req_buffer[req_cnt].request);
                         ++ req_cnt;
                         break;
                case QUIT: break;
                default: printf("INVALID SELECTION: %d \n", select);
                }
        if(select > Ø && select <= 15)
          printf("\n%s\n\n", req_buffer[(req_cnt-1)].request);
       }
}
/***************************************************************************/

man_adhoc(request)
/* This procedure allows a user to enter any type of request with delimiters */
/* This routine is useful for creating erroneous rrds requests to verify the */
/* syntax analysis capabilities of the RRDS Controller.                      */

char request[];     /* ptr to a request buffer */
{
 char c[2];             /* input character            */
 int req_index;     /* index into character array of a request */

 c[Ø] = '\Ø';
 c[1] = '\Ø';
 while (c[Ø] != BOR[Ø])
       {
        printf("Enter request with '%c', '%c' delimiters.\n", BOR[Ø], EOR[Ø]);
        prompt();
        c[Ø] = getchar();
       }
 strcpy(request, c);  /* BOR */
 req_index = 1;
 c[Ø] = getchar();
 while (c[Ø] != EOR[Ø] && req_index < (REQ_LENGTH - 1))
       {
        strcat(request, c);
        ++ req_index;
        c[Ø] = getchar();
        if (c[Ø] == '\n')
            c[Ø] = BLANK[Ø];
       }
 getchar(); /* Read \n */
 strcat(request, EOR);
}
/***************************************************************************/
```

```
man_average(request)
/* This procedure creates a rrds AVERAGE  request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                                  */

char request[];                  /* ptr to a request buffer */
{
 int at_index;                  /* attribute index in relation template    */
 int rel_index;                 /* relation index in relation template     */
 char where[MAX_WHERE];         /* store any WHERE clause                  */


/* Get relation name for the request */

printf("Generate AVERAGE request from which relation ? \n");
rel_index = get_rel();

/* Get the attribute used in the request */

printf("Generate AVERAGE request for which attribute ? \n");
at_index = 0;
while (at_index < rel_templates[rel_index].rel_nof_attrs)
      {
       if (rel_templates[rel_index].rel_attrs[at_index].at_type == 'I' ||
              rel_templates[rel_index].rel_attrs[at_index].at_type == 'i')
          {
           printf("[%d] %20s \n", at_index,
                  rel_templates[rel_index].rel_attrs[at_index].at_name);
          }
       ++ at_index;
      }
prompt();
scanf("%d", &at_index);
getchar(); /* Read \n */

/* Get the WHERE clause if necessary */

get_where_clause(rel_index, where);
```

```
/* Now create the AVERAGE request from the data provided */

strcpy(request, BOR);
strcat(request, "AVERAGE (");

/* Include the selected attribute in the request */

strcat(request, rel_templates[rel_index].rel_attrs[at_index].at_name);

/* Include the FROM clause in the request */

 strcat(request, ") FROM ");

/* Include the relation name in the request */

 strcat(request, rel_templates[rel_index].rel_name);

/* Include any WHERE clause in the request */

 if (where[0] != '\0')
    { /* Include the WHERE clause in the request */
      strcat(request, BLANK);
      strcat(request, where);
    }
 strcat(request, EOR);
 }

/******************************************************************************/

man_max(request)
/* This procedure creates a rrds MAX  request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates,                              */

char request[];                 /* ptr to a request buffer */

{
 int at_index;                  /* attribute index in relation template    */
 int rel_index;                 /* relation index in relation template     */
 char where[MAX_WHERE];         /* store any WHERE clause                  */
```

```
/* Get relation name for the request */

printf("Generate MAX request from which relation ? \n");
rel_index = get_rel();

/* Get the attribute used in the request */

printf("Generate MAX request for which attribute ? \n");
at_index = 0;
while (at_index < rel_templates[rel_index].rel_nof_attrs)
      {
       printf("[%d] %20s \n", at_index,
              rel_templates[rel_index].rel_attrs[at_index].at_name);
       ++ at_index;
      }
prompt();
scanf("%d", &at_index);
getchar(); /* Read \n */

/* Get the WHERE clause if necessary */

 get_where_clause(rel_index, where);

/* Now create the MAX request from the data provided */

 strcpy(request, BOR);
 strcat(request, "MAX (");

/* Include the selected attribute in the request */

 strcat(request, rel_templates[rel_index].rel_attrs[at_index].at_name);

/* Include the FROM clause in the request */

 strcat(request, ") FROM ");

/* Include the relation name in the request */

 strcat(request, rel_templates[rel_index].rel_name);

/* Include any WHERE clause in the request */

 if (where[0] != '\0')
    { /* Include the WHERE clause in the request */
      strcat(request, BLANK);
      strcat(request, where);
    }
 strcat(request, EOR);
 }

/***********************************************************************/
```

```
man_min(request)
/* This procedure creates a rrds MIN request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates,                           */

char request[];               /* ptr to a request buffer  */

{
 int at_index;                /* attribute index in relation template    */
 int rel_index;               /* relation index in relation template     */
 char where[MAX_WHERE];       /* store any WHERE clause                  */


/* Get relation name for the request */

printf("Generate MIN request from which relation ? \n");
rel_index = get_rel();

/* Get the attribute used in the request */

printf("Generate MIN request for which attribute ? \n");
at_index = 0;
while (at_index < rel_templates[rel_index].rel_nof_attrs)
       {
        printf("[%d] %20s \n", at_index,
                rel_templates[rel_index].rel_attrs[at_index].at_name);
        ++ at_index;
       }
prompt();
scanf("%d", &at_index);
getchar(); /* Read \n */

/* Get the WHERE clause if necessary */

 get_where_clause(rel_index, where);

/* Now create the MIN request from the data provided */

strcpy(request, BOR);
strcat(request, "MIN (");
/* Include the selected attribute in the request */

strcat(request, rel_templates[rel_index].rel_attrs[at_index].at_name);

/* Include the FROM clause in the request */

 strcat(request, ") FROM ");

/* Include the relation name in the request */

 strcat(request, rel_templates[rel_index].rel_name);

/* Include any WHERE clause in the request */

 if (where[0] != '\0')
    { /* Include the WHERE clause in the request */
      strcat(request, BLANK);
      strcat(request, where);
    }
 strcat(request, EOR);
 }

/************************************************************************/

man_sum(request)
/* This procedure creates a rrds SUM request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates,                           */

char request[];                /* ptr to a request buffer */
{
 int at_index;                 /* attribute index in relation template    */
 int rel_index;                /* relation index in relation template     */
 char where[MAX_WHERE];        /* store any WHERE clause                  */

/* Get relation name for the request */

printf("Generate SUM request from which relation ? \n");
rel_index = get_rel();
```

```
/* Get the attribute used in the request */

printf("Generate SUM request for which attribute ? \n");
at_index = 0;
while (at_index < rel_templates[rel_index].rel_nof_attrs)
      {
       if (rel_templates[rel_index].rel_attrs[at_index].at_type == 'I' ||
              rel_templates[rel_index].rel_attrs[at_index].at_type == 'i')
          {
           printf("[%d] %20s \n", at_index,
                  rel_templates[rel_index].rel_attrs[at_index].at_name);
          }
       ++ at_index;
      }
prompt();
scanf("%d", &at_index);
getchar(); /* Read \n */
/* Get the WHERE clause if necessary */

 get_where_clause(rel_index, where);

/* Now create the SUM request from the data provided */

strcpy(request, BOR);
strcat(request, "SUM (");

/* Include the selected attribute in the request */

strcat(request, rel_templates[rel_index].rel_attrs[at_index].at_name);

/* Include the FROM clause in the request */

 strcat(request, ") FROM ");

/* Include the relation name in the request */

 strcat(request, rel_templates[rel_index].rel_name);

/* Include any WHERE clause in the request */

 if(where[0] != '\0')
   { /* Include the WHERE clause in the request */
     strcat(request, BLANK);
     strcat(request, where);
   }
 strcat(request, EOR);
 }

/***************************************************************************/
```

```
man_count(request)
/* This procedure creates a rrds COUNT request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                              */

char request[];                /* ptr to a request buffer */

{
 int rel_index;                /* relation index in relation template    */
 char where[MAX_WHERE];        /* store any WHERE clause                 */


/* Get relation name for the request */

 printf("Generate COUNT request from which relation ? \n");
 rel_index = get_rel();

/* Get the WHERE clause if necessary */

 get_where_clause(rel_index, where);

/* Now create the COUNT request from the data provided */

 strcpy(request, BOR);
 strcat(request, "COUNT");

/* Include the FROM clause in the request */

 strcat(request, " FROM ");

/* Include the relation name in the request */

 strcat(request, rel_templates[rel_index].rel_name);

/* Include any WHERE clause in the request */

 if (where[0] != '\0')
    { /* Include the WHERE clause in the request */
      strcat(request, BLANK);
      strcat(request, where);
    }
 strcat(request, EOR);
 }

/*************************************************************************/
```

```
man_delete(request)
/* This procedure creates a rrds DELETE  request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates, req_cnt, req_buffer[]         */

char request[];                 /* ptr to a request buffer */
{
 int del_type;                  /* type of delete request                    */
 int rel_index;                 /* relation index in relation template       */
 char where[MAX_WHERE];         /* store any WHERE clause                    */


 /* Get relation name for the request */

 printf("Generate DELETE request from which relation ? \n");
 rel_index = get_rel();

 del_type = 0;
 while(del_type <= 0 || del_type > 2)
      {
       printf("Please choose the type of DELETE request to be created:\n\n");
       printf("[1]  DELETE ALL FROM Relation\n");
       printf("[2]  DELETE FROM Relation WHERE (Specifier)\n");
       prompt();
       scanf("%d", &del_type);
       getchar(); /* Read \n */
      }

 strcpy(request, BOR);
 if(del_type == 1)
   {
    strcat(request, "DELETE ALL FROM ");
    strcat(request, rel_templates[rel_index].rel_name);
   }
 if(del_type == 2)
   {
    /* Get the WHERE clause if necessary */
    get_where_clause(rel_index, where);
    strcat(request, "DELETE FROM ");
    strcat(request, rel_templates[rel_index].rel_name);

    /* Include any WHERE clause in the request */

    if (where[0] != '\0')
       { /* Include the WHERE clause in the request */
        strcat(request, BLANK);
        strcat(request, where);
       }
    }
 strcat(request, EOR);
 }

/***************************************************************************/
```

```
man_select(request)
/* This procedure creates a rrds  SELECT request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                                */

char request[];              /* ptr to a request buffer */
{
 int sel_type;               /* type of SELECT request                 */
 int rel_index;              /* relation index in relation template    */
 char where[MAX_WHERE];      /* store any WHERE clause                 */

 /* Get relation name for the request */

 printf("Generate SELECT request from which relation ? \n");
 rel_index = get_rel();

 sel_type = 0;
 while(sel_type <= 0 || sel_type > 2)
      {
       printf("Please choose the type of SELECT request to be created:\n\n");
       printf("[1]  SELECT ALL FROM Relation \n");
       printf("[2]  SELECT FROM Relation WHERE (Specifier) \n");
       prompt();
       scanf("%d", &sel_type);
       getchar(); /* Read \n */
      }

 strcpy(request, BOR);
 if(sel_type == 1)
   {
    strcat(request, "SELECT ALL FROM ");
    strcat(request, rel_templates[rel_index].rel_name);
   }
 if(sel_type == 2)
   {
    /* Get the WHERE clause if necessary */
    get_where_clause(rel_index, where);
    strcat(request, "SELECT FROM ");
    strcat(request, rel_templates[rel_index].rel_name);

    /* Include any WHERE clause in the request */

    if (where[0] != '\0')
       { /* Include the WHERE clause in the request */
        strcat(request, BLANK);
        strcat(request, where);
       }
   }
 strcat(request, EOR);
 }

/**************************************************************************/
```

```
man_diff(request)
/* This procedure creates a rrds DIFF request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                             */

char request[];                /* ptr to a request buffer */
{
 int rel_1;                    /* index for first relation selected       */
 int rel_2;                    /* index for second relation selected      */

/* Get relation name for the request */

 printf("Generate DIFF request from which relation ? \n");
 rel_1 = get_rel();

/* Get the second relation used in the request */

 printf("Generate DIFF request for which other relation ? \n");
 rel_2 = get_rel();

/* Now create the DIFF request from the data provided */

 strcpy(request, BOR);

/* Include the first relation name in the request */

 strcat(request, rel_templates[rel_1].rel_name);

/* Include the DIFF clause */

 strcat(request, " DIFF ");

 /* Include the second relation name in the request */

 strcat(request, rel_templates[rel_2].rel_name);
 strcat(request, EOR);
}

/*****************************************************************************/
```

```
man_intsect(request)
/* This procedure creates a rrds INTSECT request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                                */

char request[];                /* ptr to a request buffer */
{
 int rel_1;                    /* index to the first relation selected    */
 int rel_2;                    /* index to the second relation selected   */

/* Get the first relation name for the request */

 printf("Generate INTSECT request from which relation ? \n");
 rel_1 = get_rel();

/* Get the second relation name for the request */

 printf("Generate INTSECT request for which other relation ? \n");
 rel_2 = get_rel();

/* Include the first relation name in the request */

 strcpy(request, BOR);
 strcat(request, rel_templates[rel_1].rel_name);

/* Include the INTSECT clause in the request */

 strcat(request, " INTSECT ");

/* Include the second relation name in the request */

 strcat(request, rel_templates[rel_2].rel_name);
 strcat(request, EOR);
 }

/****************************************************************************/
```

```
man_join(request)
/* This procedure creates a rrds JOIN  request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                               */

char request[];                 /* ptr to a request buffer */
{
 int rel_1;                     /* index to the first relation selected    */
 int rel_2;                     /* index to the second relation selected   */

/* Get the first relation name for the request */

 printf("Generate JOIN request from which relation ? \n");
 rel_1 = get_rel();

/* Get the second relation name for the request */

 printf("Generate JOIN request for which other relation ? \n");
 rel_2 = get_rel();

/* Include the first relation name in the request */

 strcpy(request, BOR);
 strcat(request, rel_templates[rel_1].rel_name);

/* Include the JOIN clause in the request */

 strcat(request, " JOIN ");

/* Include the second relation name in the request */

 strcat(request, rel_templates[rel_2].rel_name);
 strcat(request, EOR);
 }

/***************************************************************************/
```

```
man_union(request)
/* This procedure creates a rrds UNION request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                              */

char request[];          /* ptr to a  request  buffer */
{
 int rel_1;                    /* index to the first relation selected    */
 int rel_2;                    /* index to the second relation selected   */

/* Get the first relation name for the request */

 printf("Generate UNION request from which relation ? \n");
 rel_1 = get_rel();

/* Get the second relation name for the request */

 printf("Generate UNION request for which other relation ? \n");
 rel_2 = get_rel();

/* Include the first relation name in the request */

 strcpy(request, BOR);
 strcat(request, rel_templates[rel_1].rel_name);

/* Include the UNION clause in the request */

 strcat(request, " UNION ");

/* Include the second relation name in the request */

 strcat(request, rel_templates[rel_2].rel_name);
 strcat(request, EOR);
}

/****************************************************************************/
```

```
man_insert(request)
/* This procedure creates a rrds INSERT request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                                */

char request[];                /* ptr to a request  buffer */
{
 int at_index;                 /* attribute index in relation template    */
 int rel_index;                /* relation index in relation template     */

 struct {
         char data[VAL_SIZE];
        } input[MAX_ATTRS];

/* Get relation name for the request */

printf("Generate INSERT request from which relation ? \n");
rel_index = get_rel();

/* Get data for the attributes used in the request */

printf("Enter data for each attribute.\n");
printf("NOTE: Character data must start with an upper case letter.\n");
at_index = 0;
while (at_index < rel_templates[rel_index].rel_nof_attrs)
        {
         printf("%20s ", rel_templates[rel_index].rel_attrs[at_index].at_name);
         printf("(%c) ", rel_templates[rel_index].rel_attrs[at_index].at_type);
         prompt();
         gets(input[at_index].data);
         ++ at_index;
        }

/* Now create the INSERT request from the data provided */

 strcpy(request, BOR);
 strcat(request, "INSERT (");
/* Include attribute value pairs in the request */

 at_index = 0;
 while(at_index < rel_templates[rel_index].rel_nof_attrs)
      {
       if(at_index > 0)
          {
           strcat(request, ",");
          }
       strcat(request, rel_templates[rel_index].rel_attrs[at_index].at_name);
       strcat(request, ":= ");
       strcat(request, input[at_index].data);
       ++ at_index;
      }

/* Include the INTO clause in the request */

 strcat(request, ") INTO ");

/* Include the relation name in the request */

 strcat(request, rel_templates[rel_index].rel_name);
 strcat(request, EOR);
 }

/*************************************************************************/

man_project(request)
/* This procedure creates a rrds PROJECT request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                                */

char request[];                /* ptr to a request buffer */
{
 int at_index;                 /* attribute index in relation template    */
 int at_selected[MAX_ATTRS];   /* attr selected in request                */
 int proj_index;               /* number of attributes projected          */
 int rel_index;                /* relation index in relation template     */
 char select[2];               /* store user's y/n response               */
 char where[MAX_WHERE];        /* store any WHERE clause                  */

/* Initialize array to -1, positive values determine which attr is used */
at_index = 0;
while (at_index < MAX_ATTRS)
      {
       at_selected[at_index] = -1;
       ++ at_index;
      }

/* Get relation name for the request */

printf("Generate PROJECT request from which relation ? \n");
rel_index = get_rel();
```

```
/* Get the attributes used in the request */

printf("Generate PROJECT request for which attributes ? \n");
at_index = 0;
while (at_index < rel_templates[rel_index].rel_nof_attrs)
      {
       printf("%20s ", rel_templates[rel_index].rel_attrs[at_index].at_name);
       printf("  (y/n)\n");
       prompt();
       gets(select);
       if (select[0] == 'y' || select[0] == 'Y')
           at_selected[at_index] = at_index;
       ++ at_index;
      }

/* Get the WHERE clause if necessary */

 get_where_clause(rel_index, where);

/* Now create the PROJECT request from the data provided */

 strcpy(request, BOR);
 strcat(request, "PROJECT (");

/* Include any selected attributes in the request */

 proj_index = 0;
 at_index = 0;
 while(at_index < rel_templates[rel_index].rel_nof_attrs)
     {
      if(at_selected[at_index] >= 0)
        {
         if(proj_index > 0)
           {
            strcat(request, ",");
           }
         strcat(request, rel_templates[rel_index].rel_attrs[at_index].at_name);
         ++ proj_index;
        }
      ++ at_index;
     }
/* Include the FROM clause in the request */

 strcat(request, ") FROM ");

/* Include the relation name in the request */

 strcat(request, rel_templates[rel_index].rel_name);

/* Include any WHERE clause in the request */

 if (where[0] != '\0')
    { /* Include the WHERE clause in the request */
      strcat(request, BLANK);
      strcat(request, where);
    }
 strcat(request, EOR);
}

/*****************************************************************************/

man_update(request)
/* This procedure creates an rrds UPDATE request  with user input */
/* GLOBAL: tmpl_cnt, rel_templates                                */

char request[];               /* ptr to a request  buffer */
{
 int at_index;                /* attribute index in relation template    */
 int first_pair;              /* test for first attr-value pair          */
 int rel_index;               /* relation index in relation template     */
 char where[MAX_WHERE];       /* store any WHERE clause                  */

 struct {
        char data[VAL_SIZE];
       } input[MAX_ATTRS];

/* Get relation name for the request */

printf("Generate UPDATE request from which relation ? \n");
rel_index = get_rel();

/* Get data for the attributes used in the request */

printf("Enter new data for each attribute or hit 'return' for no change. \n");
printf("NOTE: Character data must start with an upper case letter.\n");
at_index = 0;
while (at_index < rel_templates[rel_index].rel_nof_attrs)
      {
       printf("%20s ", rel_templates[rel_index].rel_attrs[at_index].at_name);
       printf("(%c) ", rel_templates[rel_index].rel_attrs[at_index].at_type);
       prompt();
       gets(input[at_index].data);
       ++ at_index;
      }
```

```
/* Get the WHERE clause if applicable */

 get_where_clause(rel_index, where);

/* Now create the UPDATE request from the data provided */
 strcpy(request, BOR);
 strcat(request, "UPDATE (");


/* Include attribute value pairs in the request */

 at_index = 0;
 first_pair = 0;

 while(at_index < rel_templates[rel_index].rel_nof_attrs)
      {
       if(input[at_index].data[0] != '\0')
          {
           if(first_pair > 0)
              {
               strcat(request, ",");
              }
           strcat(request, rel_templates[rel_index].rel_attrs[at_index].at_name);
           strcat(request, " := ");
           strcat(request, input[at_index].data);
           ++ first_pair;
          }
       ++ at_index;
      }

/* Include the IN clause in the request */

 strcat(request, ") IN ");

/* Include the relation name in the request */

 strcat(request, rel_templates[rel_index].rel_name);

/* Include any WHERE clause in the request */

 if(where[0] != '\0')
    {
     strcat(request, BLANK);
     strcat(request, where);
    }
 strcat(request, EOR);
 }
/*******************************************************************************/
```

```
edit_req_file()
/* This procedure allows a user to edit a specific rrds request file. */
/* A menu of editing options is displayed and executed. Changes can   */
/* be saved if specified by the user                                  */
/* GLOBAL:  req_cnt, req_buffer[]                                     */
{
int index;   /* counter */
int selection;
char save[2];

printf("Enter the name of a request file to be edited \n");
prompt();
gets(file_name);
file_desc = fopen(file_name, "r");
while (file_desc == NULL)
      {
       printf("ERROR: Cannot open file %s \n", file_name);
       printf("Enter a valid file name or hit 'return' key \n");
       prompt();
       gets(file_name);
       if (file_name[Ø] == '\Ø')
          return;
       file_desc = fopen(file_name, "r");
      }

req_cnt = Ø;
while(fgets(req_buffer[req_cnt].request, REQ_LENGTH, file_desc) != NULL)
     {
       /* Remove line feed character from request */
       index = strindex(&req_buffer[req_cnt].request[Ø], "\n");
       if (index >= Ø)
          {
           req_buffer[req_cnt].request[index] = '\Ø';
          }
       ++ req_cnt;
     }
fclose(file_desc);
selection = 1;
while (selection != QUIT)
      {
       printf("Enter type of editing to be performed: \n");
       printf("[1] DISPLAY Requests \n");
       printf("[2] DELETE  Requests \n");
       printf("[3] ADD     Requests \n");
       printf("[4] MODIFY  Requests \n");
       printf("[%d] QUIT    Editor   \n",QUIT);
       prompt();
       scanf("%d", &selection);
       getchar(); /* Read \n */
       switch(selection)
             {
              case 1: display_req();
                      break;
              case 2: delete_req();
                      break;
              case 3: add_req();
                      break;
              case 4: modify_req();
                      break;
              case QUIT: break;
              default:  printf("INVALID SELECTION %d \n", selection);
             }
      }
printf("SAVE all changes ? \n");
prompt();
gets(save);
if (save[Ø] == 'y' || save[Ø] == 'Y')
   {/* save all changes */
    file_desc = fopen(file_name, "w");
    index = Ø;
    while ( index < req_cnt)
          {
           fprintf(file_desc, "%s\n", req_buffer[index].request);
           ++ index;
          }
    printf("Requests saved in %s. \n", file_name);
   }

fclose(file_desc);
}
/***************************************************************************/
```

```
display_req()
/* This procedure allows a user to view all requests or selected requests */
/* in a file.  Requests are displayed 5  at a time when all requests must */
/* be viewed.                                                              */
/* GLOBAL:  req_cnt, req_buffer[]                                          */
{
 int count;           /* count up to 5 requests then reset      */
 char mode[2];        /* a/s to display all or specific requests */
 int req_index;       /* index to req_buffer                     */
 int total_count;     /* total number of requests displayed      */
 char select[2];          /* display request specified by user      */

 mode[0] = '0';
 while (mode[0] != 'q' && mode[0] != 'Q')
        {
         printf("Enter a/s/q to display all, specific, or quit.\n");
         prompt();
         gets(mode);
         if(mode[0] == 'a' || mode[0] == 'A')
           {
            total_count = 0;
            select[0] = '0';
            while(select[0] != 'q' && select[0] != 'Q')
                 {
                  count = 0;
                  while(count < 5 && total_count < req_cnt)
                       {
                        printf("[%d] ", total_count);
                        printf("%s\n", req_buffer[total_count].request);
                        ++ count;
                        ++ total_count;
                       }
                  printf("Enter any key to continue or 'q' to quit.\n");
                  prompt();
                  gets(select);
                 }
           }
         else if(mode[0] == 's' || mode[0] == 'S')
                {
                 printf("Enter the index of request to display (0 - %d).\n",
                       (req_cnt - 1));
                 prompt();
                 scanf("%d", &req_index);
                 getchar();    /* Read '\n' */
                 if(req_index >= 0 && req_index < req_cnt)
                   {
                    printf("[%d] ", req_index);
                    printf("%s\n", req_buffer[req_index].request);
                   }
                  else if(req_index < 0 || req_index >= req_cnt)
                          printf("INVALID INDEX \n");
                 }
        }
}
/***************************************************************************/
```

```
delete_req()
/* This procedure allows a user to delete requests  from a file. */
/* The user is prompted for the request to be deleted.            */
/* GLOBAL:  req_cnt, req_buffer[]                                 */
{
 int index;          /* index counter for request array   */
 char mode[2];       /* quit or continue                  */
 int next;           /* select + 1                        */
 int select;         /* location of request to be deleted */
 char y_n[2];            /* user's y/n response to delete     */

 index = 0;
 while(index < req_cnt)
      {
       printf("[%d]  %s\n", index, req_buffer[index].request);
       ++ index;
      }

 mode[0] = '0';
 while(mode[0] != 'n' && mode[0] != 'N')
      {
       select = -1;
       while(select < 0 || select >= req_cnt)
            {
             printf("Enter position of request to be deleted (0 - %d).\n",
                    (req_cnt - 1));
             prompt();
             scanf("%d", &select);
             getchar(); /* Read \n */
            }
       printf("%s\n", req_buffer[select].request);
       printf("DELETE this request ? \n");
       prompt();
       gets(y_n);
       if(y_n[0] == 'y' || y_n[0] == 'Y')
         {
          next = select + 1;
          while(next < req_cnt)
               {
                strcpy(req_buffer[(next - 1)].request,
                       req_buffer[next].request);
                ++ next;
               }
          -- req_cnt;
         }
      printf("Delete another request ? (y/n). \n");
      prompt();
      gets(mode);
     }
 }
/*****************************************************************************/
```

```
add_req()
/* This procedure allows the user to insert specific type of rrds requests  */
/* at a specific location in a request file.  A menu of request types is    */
/* displayed and created.                                                   */
/* GLOBAL:  req_cnt, req_buffer[]                                           */
{
 int index;        /* index into request character array */
 int next;         /* after adjusting req_buffer, next = insert position */
 int position;     /* actual position to insert into req_buffer */
 int select;       /* indicates request type to be inserted     */

 select = 1;
 if (req_cnt >= MAX_REQ)
    {
    printf("WARNING:  This file contains the maximum number of requests.\n");
    return;
    }
 while (select != QUIT && req_cnt < MAX_REQ)
      {
       printf("Select the type of request to be inserted. \n");
       printf("[1] AVERAGE    [5] INSERT    [9]  MIN       [13] UNION   \n");
       printf("[2] COUNT      [6] INTSECT   [10] PROJECT   [14] UPDATE \n");
       printf("[3] DELETE     [7] JOIN      [11] SELECT    [15] ADHOC/ERROR\n");
       printf("[4] DIFF       [8] MAX       [12] SUM       [%d] QUIT \n",QUIT);
       prompt();
       scanf("%d", &select);
       getchar(); /* Read \n */
       if(select != QUIT)
         {
          position = -1;
          while(position < 0 || position > req_cnt)
               {
                printf("Enter file position for request insertion (0 - %d).\n",
                      req_cnt);
                prompt();
                scanf("%d", &position);
                getchar(); /* Read \n */
               }
          /* Make room in req_buffer for the next new request. */
          next = req_cnt;
          while (next > position)
                {
                 strcpy(req_buffer[next].request,
                        req_buffer[(next - 1)].request);
                 -- next;
                }
         }
```

```
        switch (select)
               {
               case 1:  man_average(req_buffer[position].request);
                        ++ req_cnt;
                        break;
               case 2:  man_count(req_buffer[position].request);
                        ++ req_cnt;
                        break;
               case 3:  man_delete(req_buffer[position].request);
                        ++ req_cnt;
                        break;
               case 4:  man_diff(req_buffer[position].request);
                        ++ req_cnt;
                        break;
               case 5:  man_insert(req_buffer[position].request);
                        ++ req_cnt;
                        break;
               case 6:  man_intsect(req_buffer[position].request);
                        ++ req_cnt;
                        break;
               case 7:  man_join(req_buffer[position].request);
                        ++ req_cnt;
                        break;
               case 8:  man_max(req_buffer[position].request);
                        ++ req_cnt;
                        break;
               case 9:  man_min(req_buffer[position].request);
                        ++ req_cnt;
                        break;
               case 10: man_project(req_buffer[position].request);
                        ++ req_cnt;
                        break;
               case 11: man_select(req_buffer[position].request);
                        ++ req_cnt;
                        break;
               case 12: man_sum(req_buffer[position].request);
                        ++ req_cnt;
                        break;
               case 13: man_union(req_buffer[position].request);
                        ++ req_cnt;
                        break;
               case 14: man_update(req_buffer[position].request);
                        ++ req_cnt;
                        break;
               case 15: man_adhoc(req_buffer[position].request);
                        ++ req_cnt;
                        break;
               case QUIT: break;
               default: printf("INVALID SELECTION: %d \n", select);
               }
        if(select > 0 && select <= 15)
          printf("\n%s\n\n", req_buffer[position].request);
        if(req_cnt == MAX_REQ)
          printf("WARNING:  Maximum number of requests now exist in file.\n");
       } /* end of while select != quit */
   }
   /*************************************************************************/
```

```
modify_req()
/* This procedure allows a user to modify  specific requests from a file.  */
/* The user is prompted for the character string to be modified and the   */
/* replacement string.                                                     */
/* GLOBAL:  req_cnt, req_buffer[]                                          */
{
 int count;                     /* position counter in character array    */
 int find;                      /* result of strindex function            */
 int index;                     /* index into character array             */
 int max;                       /* exclusive upper limit of a string index */
 int mod_index;                 /* index for new_string array             */
 char mode[2];                  /* quit or continue processing            */
 int new_length;                /* length of new_string                   */
 int old_length;                /* length of old string                   */
 int position;                  /* position of request in req_buffer      */
 int end_shift;                 /* shift boundary limit                   */
 int l_shift;                   /* number of left shift positions         */
 int r_shift;                   /* number of right shift positions        */
 char y_n[2];                   /* user's response                        */
 char old_string[REQ_LENGTH];
 char new_string[REQ_LENGTH];

 mode[0] = '0';
 while(mode[0] != 'n' && mode[0] != 'N')
      {
       position = -1;
       while(position < 0 || position >= req_cnt)
            {
             printf("Enter position of request to be modified (0 - %d).\n",
                  (req_cnt - 1));
             prompt();
             scanf("%d", &position);
             getchar(); /* Read \n */
            }
       printf("%s\n", req_buffer[position].request);
       printf("Modify this request ?  (y/n).\n");
       prompt();
       gets(y_n);
       if(y_n[0] == 'y' || y_n[0] == 'Y')
         {
          printf("Enter the string to be changed. \n");
          prompt();
          gets(old_string);
          printf("Enter the new string. \n");
          prompt();
          gets(new_string);
          find = strindex(&req_buffer[position].request[0], &old_string[0]);
          if (find >= 0)
             {
              old_length = strlen(old_string);
              new_length = strlen(new_string);
```

```
if (new_length == old_length)
   {
    index = find;
    mod_index = 0;
    max = find + new_length;
    while(index < max)
         {
          req_buffer[position].request[index] =
                             new_string[mod_index];
          ++ index;
          ++ mod_index;
         }
   }
else if (new_length < old_length)
   {
    /* fill with new string and left shift the request  */
    index = find;
    mod_index = 0;
    max = find + new_length;
    while(index < max)
         {
          req_buffer[position].request[index] =
                     new_string[mod_index];
          ++ index;
          ++ mod_index;
         }
    l_shift = old_length - new_length;
    max = strlen(req_buffer[position].request);
    max = max - l_shift;
    while(index <= max)
         {
          req_buffer[position].request[index] =
              req_buffer[position].request[(index + l_shift)];
          ++ index;
         }
   }
```

```
          else if (new_length > old_length)
            {
             /* right shift the request then fill it with new data */
             r_shift = new_length - old_length;
             max = strlen(req_buffer[position].request);
             max = max + r_shift;
             if (max > REQ_LENGTH)
                 max = REQ_LENGTH;
             end_shift = find + old_length + r_shift;
             count = max;
             while(count >= end_shift)
                  {
                   req_buffer[position].request[count] =
                       req_buffer[position].request[(count - r_shift)];
                   -- count;
                  }
             /* Now fill request with new string */
             index = find;
             mod_index = 0;
             max = find + new_length;
             while(index < max)
                  {
                   req_buffer[position].request[index] =
                                      new_string[mod_index];
                   ++ index;
                   ++ mod_index;
                  }
            }
          }
      else if (find < 0)
             printf("String not found:  %s\n", old_string);
     }
    printf("%s\n", req_buffer[position].request);
    printf("Modify another request ? (y/n). \n");
    prompt();
    gets(mode);
   }
}

/****************************************************************************/
```

```
run_req_file()
/* This procedure allows a user to execute requests.                          */
/* The requests are passed to the Controller by the socket facility of VAX.  */
/* The channel 'busl' is created and used to pass either all the requests    */
/* or selected requests to the Controller. The results can be displayed on   */
/* a screen and stored into a specific output file.                          */
/* GLOBAL:  active_bus, req_cnt, req_buffer[], output                        */

{
 char mode[2];
 int index;

 if (active_bus == Ø)
     {
      connect_bus();
      active_bus = 1;
     }

 printf("Enter name of request file to be used for processing requests. \n");
 prompt();
 gets(file_name);
 file_desc = fopen(file_name, "r");
 while (file_desc == NULL)
        {
         printf("ERROR:  Cannot open file %s \n", file_name);
         printf("Enter valid file name or hit 'RETURN' key to exit \n");
         prompt();
         gets(file_name);
         if (file_name[Ø] == '\n')
            return;
         file_desc = fopen(file_name, "r");
        }
 req_cnt = Ø;
 while (fgets(req_buffer[req_cnt].request, REQ_LENGTH, file_desc) != NULL)
        {
         /* Remove line feed character from the request */
         index = strindex(&req_buffer[req_cnt].request[Ø], "\n");
         if (index >= Ø)
            {
             req_buffer[req_cnt].request[index] = '\Ø';
            }
         ++ req_cnt;
        }
 fclose(file_desc);
output[Ø] = 'Ø';
while (output[Ø] != 's' && output[Ø] != 'f' && output[Ø] != 'b')
        {
         printf("Enter s/f/b to direct output to a screen, file, or both.\n");
         prompt();
         gets(output);
        }
if (output[Ø] == 'f' || output[Ø] == 'b')
    {
     printf("Enter name of output file.\n");
     prompt();
     gets(out_file);
     out_desc = fopen(out_file, "w");
     while(out_desc == NULL)
           {
            printf("ERROR:  Cannot open output file %s \n", out_file);
            printf("Enter name of output file. \n");
            prompt();
            gets(out_file);
            out_desc = fopen(out_file, "w");
           }
    }
mode[Ø] = 'Ø';
while(mode[Ø] != 'q' && mode[Ø] != 'Q')
      {
       printf("Enter a/s/q to run all requests, selected requests, or quit\n");
       prompt();
       gets(mode);
       if (mode[Ø] == 'a' || mode[Ø] == 'A')
          run_all();
       else if (mode[Ø] == 's' || mode[Ø] == 'S')
               run_selected();
      }

if (output[Ø] == 'f' || output[Ø] == 'b')
   fclose(out_desc);
}

/****************************************************************************/
```

```
run_all()
/* This procedure allows all rrds requests in the req_buffer to be    */
/* processed by the Controller. Requests are sent via channel 'busl'. */
/* GLOBAL:  req_cnt, req_buffer, output, channel                      */
{
 int index;                  /* index into req_buffer                 */
 int err;                    /* error code from send and recv         */

 index = Ø;
 while(index < req_cnt)
      {
       if(output[Ø] == 'b' || output[Ø] == 's')
          {
           printf("Sending:\n");
           printf("%s\n", req_buffer[index].request);
          }
       err = send(channel, req_buffer[index].request, REQ_LENGTH, Ø);
       if (err < Ø)
           {
            printf("ERROR:  send command failed.\n");
            return;
           }
       err = recv(channel, reply, RES_SIZE, Ø);
       if (err < Ø)
           {
            printf("ERROR:  recv command failed.\n");
            return;
           }

       if(reply[Ø] == CONTINUE[Ø])
          {
           if (output[Ø] == 'b' || output[Ø] == 'f')
              {
               fprintf(out_desc, "\n%s\n", "SENDING: ");
               fprintf(out_desc, "%s\n", req_buffer[index].request);
               fprintf(out_desc, "%s\n", "RESULT:  ");
              }
           while(reply[Ø] != TERMINATE[Ø])
                {
                 if(output[Ø] == 'b' || output[Ø] == 'f')
                    fprintf(out_desc, "%s\n", reply);
                 if(output[Ø] == 'b' || output[Ø] == 's')
                    printf("%s\n", reply);
                 recv(channel, reply, RES_SIZE, Ø);
                }
          }
```

```
        else /* result has no continuation symbol */
          {
           if (output[0] == 'b' || output[0] == 's')
               printf("%s\n", reply);
           if (output[0] == 'b' || output[0] == 'f')
               {
               fprintf(out_desc, "\n%s\n", "SENDING: ");
               fprintf(out_desc, "%s\n", req_buffer[index].request);
               fprintf(out_desc, "%s\n", "RESULT: ");
               fprintf(out_desc, "%s\n", reply);
               }
          }
        ++ index;
       }
  }

/*****************************************************************************/

run_selected()
/* This procedure enables users to run  specific requests in the */
/* req_buffer or interactively create and run  requests.         */
/* GLOBAL:  req_cnt, req_buffer[], output, channel               */
{
 int err;                     /* error code from send and recv      */
 int mode;                    /* quit or continue                   */
 int position;                /* buffer position of selected request*/
 char select[2];              /* s/t, selected or terminal input    */
 int type;                    /* type of request to be created & run*/
 char y_n[2];                 /* user's response                    */
 char t_request[REQ_LENGTH]; /* request buffer for terminal input*/


 display_req();
 printf("Enter \n");
 printf("  [f]: to run a request from the file already selected\n");
 printf("  [t]: to run a request from the terminal\n");
 printf("  [q]: to quit\n");
 prompt();
 gets(select);
 while (select[0] != 'q' && select[0] != 'Q')
       {
        if(select[0] == 'f' || select[0] == 'F')
          {
           position = -1;
           while(position < 0 || position >= req_cnt)
                {
                 printf("Enter position of request to be processed (0 - %d).\n",
                        (req_cnt - 1));
                 prompt();
                 scanf("%d", &position);
                 getchar(); /* Read \n */
                }
           printf("%s\n", req_buffer[position].request);
           printf("Run this request ?  (y/n) \n");
           prompt();
           gets(y_n);
        if(y_n[0] == 'y' || y_n[0] == 'Y')
          {
           if(output[0] == 'b' || output[0] == 's')
             {
              printf("Sending:\n");
              printf("%s\n", req_buffer[position].request);
             }
           err = send(channel, req_buffer[position].request, REQ_LENGTH, 0);
           if(err < 0)
             {
              printf("ERROR:  send command failed.\n");
              return;
             }
           err = recv(channel, reply, RES_SIZE, 0);
           if(err < 0)
             {
              printf("ERROR:  recv command failed. \n");
              return;
             }
           if(reply[0] == CONTINUE[0])
             {
              if(output[0] == 'b' || output[0] == 'f')
                {
                 fprintf(out_desc, "\n%s\n", "SENDING:  ");
                 fprintf(out_desc, "%s\n",req_buffer[position].request);
                 fprintf(out_desc, "%s\n", "RESULT:  ");
                }
              while(reply[0] != TERMINATE[0])
                  {
                   if(output[0] == 'b' || output[0] == 'f')
                      fprintf(out_desc, "%s\n", reply);
                   if(output[0] == 'b' || output[0] == 's')
                      printf("%s\n", reply);
                   recv(channel, reply, RES_SIZE, 0);
                  }
             }
           else /* result has no continuation symbol */
             {
              if(output[0] == 'b' || output[0] == 's')
                 printf("%s\n", reply);
              if(output[0] == 'b' || output[0] == 'f')
                {
                 fprintf(out_desc, "\n%s\n", "SENDING:");
                 fprintf(out_desc, "%s\n", req_buffer[position].request);
                 fprintf(out_desc, "%s\n", "RESULT:");
                 fprintf(out_desc, "%s\n", reply);
                }
              }
          }
  } /* end if (select = file) */
```

```
if(select[0] == 't' || select[0] == 'T')
  {
   type = -1;
   while(type < 0 || type > 15)
     {
      printf("Which request do you want to run ?\n");
      printf("[1] AVERAGE    [5] INSERT    [9]  MIN      [13] UNION  \n");
      printf("[2] COUNT      [6] INTSECT   [10] PROJECT  [14] UPDATE \n");
      printf("[3] DELETE     [7] JOIN      [11] SELECT   [15] ADHOC/ERROR\n");
      printf("[4] DIFF       [8] MAX       [12] SUM      [%d] QUIT \n",QUIT);

      prompt();
      scanf("%d", &type);
      getchar(); /* Read \n */
     }
   switch (type)
         {
          case 1:  man_average(t_request);
                   break;
          case 2:  man_count(t_request);
                   break;
          case 3:  man_delete(t_request);
                   break;
          case 4:  man_diff(t_request);
                   break;
          case 5:  man_insert(t_request);
                   break;
          case 6:  man_intsect(t_request);
                   break;
          case 7:  man_join(t_request);
                   break;
          case 8:  man_max(t_request);
                   break;
          case 9:  man_min(t_request);
                   break;
          case 10: man_project(t_request);
                   break;
          case 11: man_select(t_request);
                   break;
          case 12: man_sum(t_request);
                   break;
          case 13: man_union(t_request);
                   break;
          case 14: man_update(t_request);
                   break;
          case 15: man_adhoc(t_request);
                   break;
          case QUIT: break;
          default: printf("INVALID SELECTION: %d \n", type);
         }
      if(type < 16 && type != QUIT)
        {
          if(output[Ø] == 'b' || output[Ø] == 's')
            {
             printf("SENDING:\n");
             printf("%s\n", t_request);
            }
          err = send(channel, t_request, REQ_LENGTH, Ø);
          if(err < Ø)
            {
             printf("ERROR:  send command failed.\n");
             return;
            }
          err = recv(channel, reply, RES_SIZE, Ø);
          if(err < Ø)
            {
             printf("ERROR:  recv command failed.\n");
             return;
            }
          if(reply[Ø] == CONTINUE[Ø])
            {
             if(output[Ø] == 'b' || output[Ø] == 'f')
               {
                fprintf(out_desc, "\n%s\n", "SENDING:");
                fprintf(out_desc, "%s\n", t_request);
                fprintf(out_desc, "%s\n", "RESULT:  ");
               }
             while(reply[Ø] != TERMINATE[Ø])
                 {
                  if(output[Ø] == 'b' || output[Ø] == 'f')
                     fprintf(out_desc, "%s\n", reply);
                  if(output[Ø] == 'b' || output[Ø] == 's')
                     printf("%s\n", reply);
                  recv(channel, reply, RES_SIZE, Ø);
                 }
            }
          else /* result has no continuation symbol */
            {
             if(output[Ø] == 'b' || output[Ø] == 's')
                printf("%s\n", reply);
             if(output[Ø] == 'b' || output[Ø] == 'f')
               {
                fprintf(out_desc, "\n%s\n", "SENDING:");
                fprintf(out_desc, "%s\n", t_request);
                fprintf(out_desc, "%s\n", "RESULT:");
                fprintf(out_desc, "%s\n", reply);
               }
            }
         }
      } /* end if (select =  terminal) */
     printf("Enter \n");
     printf("  [f]:  to run a request from the file already selected\n");
     printf("  [t]:  to run a request from the terminal\n");
     printf("  [q]:  to quit\n");
     prompt();
     gets(select);
    }/* end while (more requests to run) */
   }
  /*************************************************************************/
```

```
connect_bus()  /* creates a channel 'busl' for host<->controller messages  */
{
 int  err;

/*****************  create the channel for communication ***************/

/* Get a UNIX socket channel */

 err = (make_channel = socket(AF_UNIX,SOCK_STREAM,Ø));
 if (err <= Ø)
    {
     printf("ERROR: socket command failed\n");
     exit();
    }

 /* Bind the channel with the file "busl" */

 err = bind(make_channel,&soc_name,6);
 if (err < Ø)
    {
     printf("ERROR: bind command failed\n");
     printf("Enter 'rm busl', then 'host.out' to restart\n");
     exit();
    }

/* Wait & Listen for Controller to connect to the channel */

 printf("Please login and invoke 'rrds.out' on another terminal.\n");
 err = listen(make_channel,1);
 if (err < Ø)
    {
     printf("ERROR: listen command failed\n");
     exit();
    }

/* Get location ID of Controller process */

 err = (channel = accept(make_channel,Ø,Ø));
 if (err < Ø)
    {
     printf("ERROR: accept command failed\n")·
     exit();
    }
}

/*************************************************************************/

disconnect_bus()
{
 send (channel, TERMINATE, 2, Ø);
 close(channel);
 unlink("busl");
}

/*************************************************************************/
```

```
gendigits(result, lower_b, upper_b)
/* This procedure creates a string of digits based on a bounded range */
char *result, *lower_b, *upper_b;
{
 int diff;      /* difference between the bounds          */
 int digit;     /* integer -> string digit                */
 int fwd_index; /* forward index of string                */
 int rev_index; /* reverse index of a string              */
 int high;      /* integer value of upper bound string */
 int low;       /* integer value of lower bound string */
 int genval;    /* randomly generated integer value    */

 low    = atoi(lower_b);
 high   = atoi(upper_b);
 diff   = high - low;
 genval = low + random(diff);

/* convert genval number to a string, store in result */

 rev_index = (VAL_SIZE - 1);
 *(result + rev_index) = '\Ø';
 -- rev_index;
 if(genval > Ø)
        {
         while(genval > Ø && rev_index >= Ø)
               {
                digit = genval % 1Ø;
                *(result + rev_index) =  digit + 'Ø';
                genval = genval / 1Ø;
                -- rev_index;
               }
        }
 else if (genval == Ø)
        {
         *(result + rev_index) = 'Ø';
         -- rev_index;
        }
 else if (genval < Ø)
        {
         genval = genval * -1;
         while(genval > Ø && rev_index >= Ø)
               {
                digit = genval % 1Ø;
                *(result + rev_index) = digit + 'Ø';
                genval = genval / 1Ø;
                -- rev_index;
               }
         *(result + rev_index) = '-';
         -- rev_index;
        }
 /* fill any remaining space with blanks */

 while(rev_index >= 0)
       {
        *(result + rev_index) = BLANK[0];
        -- rev_index;
       }

 /* strip off leading blanks */

 while( *(result) == BLANK[0])
       {
        for(fwd_index = 0; fwd_index < (VAL_SIZE - 1); ++ fwd_index)
            *(result + fwd_index) = *(result + fwd_index + 1);
       }
}

/***************************************************************************/

get_rel()  /* returns index to relation that is selected by the user */
{
 int index;
 for(index = 0; index < tmpl_cnt; ++ index)
     printf("[%d] %s\n", index, rel_templates[index].rel_name);
 prompt();
 scanf("%d", &index);
 getchar();  /* Read '\n' */
 return(index);
}

/****************************************************************************/

get_where_clause(rel_index, clause)
int  rel_index;
char clause[];
{
 char more_conj[2];         /* Store response for another conjunction */
 char more_pred[2];         /* Store response for another predicate   */
 char value[VAL_SIZE];      /* Store predicate value                  */
 int  at_index;             /* attribute index for a relation         */
 int  conj_cnt;             /* number of conjunctions                 */
 int  op_index;             /* index for relational operator type     */
 int  pred_cnt;             /* number of predicates                   */


 clause[0] = '\0';
 printf("Do you want to include a WHERE clause ?  (y/n)\n");
 prompt();
 gets(more_conj);
 if (more_conj[0] == 'n' || more_conj[0] == 'N')
    return;
 strcat(clause, "WHERE (");
conj_cnt = 0;
while(more_conj[0] == 'y' || more_conj[0] == 'Y')
   {
    if (conj_cnt > 0)
       strcat(clause, " OR ");
    more_pred[0] = 'y';
    pred_cnt = 0;
    strcat(clause, "(");
    while(more_pred[0] == 'y' || more_pred[0] == 'Y')
       {
        if (pred_cnt > 0)
           strcat(clause, " AND ");
        printf("Which attribute do you want in the predicate ?\n");
        for (at_index = 0; at_index < rel_templates[rel_index].rel_nof_attrs;
             ++ at_index)
             printf("[%d]  %s \n", at_index,
                   rel_templates[rel_index].rel_attrs[at_index].at_name);
        prompt();
        scanf("%d", &at_index);
        getchar();  /* Read '\n' */
        printf("Which relational operator do you want in the predicate ?\n");
        for (op_index = 0; op_index < 6; ++ op_index)
            printf("[%d]  %s \n", op_index, REL[op_index].OP);
        prompt();
        scanf("%d", &op_index);
        getchar();   /* READ '\n' */
        printf("Enter a value for the predicate\n");
        prompt();
        gets(value);
        /*** Now create the predicate expression ***/
        strcat(clause, rel_templates[rel_index].rel_attrs[at_index].at_name);
        strcat(clause, BLANK);
        strcat(clause, REL[op_index].OP);
        strcat(clause, BLANK);
        strcat(clause, value);
        ++ pred_cnt;
        printf("Do you want to include another predicate ?\n");
        prompt();
        gets(more_pred);
       }/* end while (more predicates) */
    strcat(clause, ")");
    ++ conj_cnt;
    printf("Do you want to include another conjunction ?\n");
    prompt();
    gets(more_conj);
   }/* end while (more conjunctions) */
if (conj_cnt > 0)
   strcat(clause, ")");
}

/***************************************************************************/
```

```
prompt()
{
 printf("->");
}

/************************************************************************/

show_req_file()
/* This procedure displays the rrds stored in a request file */
{
 int index;

 printf("Enter the name of a request file to be displayed \n");
 prompt();
 gets(file_name);
 file_desc = fopen(file_name, "r");
 while (file_desc == NULL)
        {
         printf("ERROR: Cannot open file %s \n", file_name);
         printf("Enter a valid file name or hit 'return' key \n");
         prompt();
         gets(file_name);
         if (file_name[0] == '\0')
            return;
         file_desc = fopen(file_name, "r");
        }

 req_cnt = 0;
 while(fgets(req_buffer[req_cnt].request, REQ_LENGTH, file_desc) != NULL)
       {
         /* Remove line feed character from request */
         index = strindex(&req_buffer[req_cnt].request[0], "\n");
         if (index >= 0)
            {
             req_buffer[req_cnt].request[index] = '\0';
            }
         ++ req_cnt;
       }
 fclose(file_desc);
 display_req();
}

/************************************************************************/
```

```
show_templates()
/* This procedure displays relation templates in the */
/* 'rel_templates' buffer one at a time.             */
/* GLOBAL: rel_templates[], tmpl_cnt                 */
{
 int at_pos;              /* attribute index */
 int rel_index;           /* relation index  */
 char select[2];          /* user's response */

 rel_index = Ø;
 while(rel_index < tmpl_cnt)
       {
        printf("RELATION:  %s\n",rel_templates[rel_index].rel_name);
        at_pos = Ø;
        while(at_pos < rel_templates[rel_index].rel_nof_attrs)
             {
              printf("          %s",
                     rel_templates[rel_index].rel_attrs[at_pos].at_name);
              printf(" (%c)\n",
                     rel_templates[rel_index].rel_attrs[at_pos].at_type);
              ++ at_pos;
             }
        printf("More ?  (y/n) \n");
        prompt();
        gets(select);
        if ((select[Ø] == 'n') || (select[Ø] == 'N') ||
            (select[Ø] == 'q') || (select[Ø] == 'Q'))
             break;
        ++ rel_index;
       }
 }

/****************************************************************************/
```

APPENDIX I

C LANGUAGE SOURCE CODE FOR THE RRDS CONTROLLER PROGRAM

```
/*****************************************************************/
/*                                                               */
/*                    RRDS CONTROLLER                            */
/*                                                               */
/*****************************************************************/
/*                                                               */
/*                 Developed and Written by                      */
/*                     LEONARD A. LYON                           */
/*                      Spring 1987                              */
/*                                                               */
/*  This program implements the functions of the RRDS            */
/*  controller.  It accepts requests from the host               */
/*  simulator program via a socket channel created by            */
/*  host simultor program, parses the requests using a           */
/*  YACC generated parser, sends valid requests to the           */
/*  BROADCAST file (simulated bus), generates simulated          */
/*  RC result data for valid requests, coalesces the             */
/*  result data, and sends results to the host                   */
/*  simulator program via the socket channel.                    */
/*                                                               */
/*****************************************************************/


#include <ctype.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include "rrds.vars"
#include "rrds.functions"
char *yyin;          /* input pointer for LEX; modified for RRDS      */
#include "y.tab.c"   /* source code for YACC created parser; yyparse() */


FILE *broad_desc;    /* descriptor for the BROADCAST file          */
int buf_index;       /* index for RC_buffer                        */
int buf_limit;       /* actual number of coalesced data fragments  */
int data_sent;       /* counter for relation fragments sent        */
int err;             /* error value from socket routines; 0:= OK   */
int valid_req;       /* result of parser; 0 := good,  1 := bad     */

     /* Comparison buffer used for coalescing relation fragments */
char compare_buffer[MAX_DATA_LENGTH];

     /* Temporary relation buffer holding relation fragments from RCs */
struct {
        char data[MAX_DATA_LENGTH];
       }RC_buffer[(MAX_RC_DATA * RC_ACTIVE)];
```

```
/* BEGIN RRDS CONTROLLER */

main()
{

 /* Initialize string constants */

 strcpy(TERMINATE, "@");
 strcpy(CONTINUE, "+");

 /* Use the socket facility to connect the Controller to the */
 /* same interprocess channel established by the host simulator */

 err = (channel = socket(AF_UNIX,SOCK_STREAM,Ø));
 if (err <= Ø)
    {
     printf("ERROR: socket command failed\n");
     exit();
    }
 err = connect(channel,&soc_name,6);
 if (err < Ø)
    {
     printf("ERROR: connect command failed\n");
     exit();
    }

 /*  Open the BROADCAST file for simulated broadcasting of vaild requests */

 broad_desc = fopen("BROADCAST", "w");

 /* Wait for the first request from the host on the interprocess channel */

 err = recv(channel, recv_request, REQ_LENGTH, Ø);
/* Process  requests until the host terminates */

while(recv_request[0] != TERMINATE[0])
     {
      printf("%s\n", recv_request);
      yyin = recv_request;             /* Set lex input ptr */
      valid_req = yyparse();           /* Invoke the yacc parser in y.tab.c */
      if (valid_req == 0)
         {
          /* Process a valid request */

          printf("VALID REQUEST\n");

          /* Simulate broadcasting a request by writing to BROADCAST file */

          fprintf(broad_desc, "%s\n", recv_request);

          /* Simulate receiving results from RCs and coalesce results */

          get_results();
         }
      else
         {
          /* Do not process an invalid request */

          printf("INVALID REQUEST\n");

          /* Now generate error message for host */

          bad_req();
         }

      /* Transmit the results of the request to the host */

      send_results();

      /* Await the next request from the host on the interprocess channel */

      recv(channel, recv_request, REQ_LENGTH, 0);
     }
 fclose(broad_desc);
}

/*************************************************************************/
```

```
get_results()
{
 if (strindex(&recv_request[Ø], "DELETE") >= Ø)
     strcpy(reply, "VALID DELETE COMPLETE");
 else if (strindex(&recv_request[Ø], "INSERT") >= Ø)
         strcpy(reply, "VALID INSERT COMPLETE");
 else if (strindex(&recv_request[Ø], "UPDATE") >= Ø)
         strcpy(reply, "VALID UPDATE COMPLETE");
 else if (strindex(&recv_request[Ø], "AVERAGE") >= Ø)
         co_AVG();
 else if (strindex(&recv_request[Ø], "COUNT") >= Ø)
         co_COUNT();
 else if (strindex(&recv_request[Ø], "MAX") >= Ø)
         co_MAX();
 else if (strindex(&recv_request[Ø], "MIN") >= Ø)
         co_MIN();
 else if (strindex(&recv_request[Ø], "SUM") >= Ø)
         co_SUM();
 else if (strindex(&recv_request[Ø], "PROJECT") >= Ø)
         co_RECORDS();
 else if (strindex(&recv_request[Ø], "SELECT") >= Ø)
         co_RECORDS();
 else if (strindex(&recv_request[Ø], "DIFF") >= Ø)
         co_RECORDS();
 else if (strindex(&recv_request[Ø], "INTSECT") >= Ø)
         co_RECORDS();
 else if (strindex(&recv_request[Ø], "JOIN") >= Ø)
         co_RECORDS();
 else if (strindex(&recv_request[Ø], "UNION") >= Ø)
         co_RECORDS();
}

/******************************************************************************/
```

```
open_RC_files()
{
 RC_desc[0] = fopen("RC_0.data", "r");
 RC_desc[1] = fopen("RC_1.data", "r");
 RC_desc[2] = fopen("RC_2.data", "r");
 RC_desc[3] = fopen("RC_3.data", "r");
}

/****************************************************************************/

close_RC_files()
{
 int RC_index;
 for(RC_index = 0; RC_index < RC_ACTIVE; ++RC_index)
      fclose(RC_desc[RC_index]);
}

/****************************************************************************/
```

```
co_AVG()
/* Coalesce the sums and counts from the RCs to create the average */
{
 double avg;
 float  count = 0;
 float  count_datum;
 float  sum = 0;
 float  sum_datum;
 int    precision;  /* decimal precision for floating point conversion */
 int    RC_index;
 char   result[VAL_SIZE];

 /* Get simulated RC data from files */

 open_RC_files();
 for(RC_index = 0; RC_index < RC_ACTIVE; ++RC_index)
     {
      fscanf(RC_desc[RC_index], "%e", &count_datum);
      fscanf(RC_desc[RC_index], "%e", &sum_datum);
      count = count + count_datum;
      sum = sum + sum_datum;
     }
 close_RC_files();

 /* Compute the true average */

 avg = sum / count;

 /* Convert the integer value of avg to a numeric string */

 itoa((int) avg, result);

 /* Generate the reply message */

 strcpy(reply, "VALID AVERAGE = ");
 strcat(reply, result);
 strcat(reply, ".");

 /* Now convert the fractional portion of avg to a numeric string  */
 /* The digits after the decimal point are converted one at a time */

 for(precision = 1; precision <= 3; ++ precision)
    {
     avg = avg - (int) avg;
     avg = avg * 10;
     itoa((int) avg, result);
     strcat(reply, result);
    }
}

/****************************************************************************/
```

```
co_COUNT()
/* Coalesce the count from the RCs */
{
 int counter = 0;
 int datum;
 int RC_index;
 char result[VAL_SIZE];

 /* Get simulated RC data from files */

 open_RC_files();
 for(RC_index = 0; RC_index < RC_ACTIVE; ++RC_index)
      {
       fscanf(RC_desc[RC_index], "%d", &datum);
       /* Compute the total count */
       counter = counter + datum;
      }
 close_RC_files();

 /* Convert the integer value of counter to a numeric string */

 itoa(counter, result);

 /* Generate the reply message */

 strcpy(reply, "VALID COUNT = ");
 strcat(reply, result);
}

/****************************************************************************/
```

```
co_MAX()
/* Coalesce the max values from the RCs */
{
 char datum[MAX_DATA_LENGTH];     /* data from files */
 int  max_number;                  /* max digit values     */
 int  number;
 char max_val[MAX_DATA_LENGTH];   /* max character values */
 int  RC_index = Ø;

 /* Get simulated RC data from files */

 open_RC_files();
 fscanf(RC_desc[RC_index], "%s", &max_val[Ø]);
 if(isdigit(max_val[Ø]) == Ø)
    {
     /* Process string data */

     for(RC_index = 1; RC_index < RC_ACTIVE; ++RC_index)
         {
          /* Coalesce the max string values */
          fscanf(RC_desc[RC_index], "%s", &datum[Ø]);
          if (strcmp(datum, max_val) > Ø)
             strcpy(max_val, datum);
         }
     close_RC_files();

     /* Generate the reply message */

     strcpy(reply, "VALID MAX = ");
     strcat(reply, max_val);
    }
 else
    {
     /* Process numeric data */

     max_number = atoi(max_val);
     for(RC_index = 1; RC_index < RC_ACTIVE; ++RC_index)
         {
          /* Coalesce the max digit values */
          fscanf(RC_desc[RC_index], "%s", &datum[Ø]);
          number = atoi(datum);
          if (number > max_number)
              max_number = number;
         }
     close_RC_files();

     /* Generate the reply message */

     itoa(max_number, max_val);
     strcpy(reply, "VALID MAX = ");
     strcat(reply, max_val);
    }
}
/**************************************************************************/
```

```
co_MIN()
/* Coalesce the min values from the RCs */
{
 char datum[MAX_DATA_LENGTH];      /* data from files */
 int  min_number;                  /* min digit values     */
 int  number;
 char min_val[MAX_DATA_LENGTH];    /* min character values */
 int  RC_index = Ø;

 /* Get simulated RC data from files */

 open_RC_files();
 fscanf(RC_desc[RC_index], "%s", &min_val[Ø]);
 if(isdigit(min_val[Ø]) == Ø)
   {
    /* Process string data */

    for(RC_index = 1; RC_index < RC_ACTIVE; ++RC_index)
       {
        /* Coalesce the min string values */
        fscanf(RC_desc[RC_index], "%s", &datum[Ø]);
        if (strcmp(datum, min_val) < Ø)
           strcpy(min_val, datum);
       }
    close_RC_files();

    /* Generate the reply message */

    strcpy(reply, "VALID MIN = ");
    strcat(reply, min_val);
   }
 else
   {
    /* Process numeric data */

    min_number = atoi(min_val);
    for(RC_index = 1; RC_index < RC_ACTIVE; ++RC_index)
       {
        /* Coalesce the min digit values */
        fscanf(RC_desc[RC_index], "%s", &datum[Ø]);
        number = atoi(datum);
        if (number < min_number)
            min_number = number;
       }
    close_RC_files();

    /* Generate the reply message */

    itoa(min_number, min_val);
    strcpy(reply, "VALID MIN = ");
    strcat(reply, min_val);
   }
}
/**************************************************************************/
```

```
co_SUM()
/* Coalesce the sum values from the RCs */
{
 int datum;
 int RC_index;
 int sum = Ø;
 char result[VAL_SIZE];

 /* Get simulated RC data from files */

 open_RC_files();
 for(RC_index = Ø; RC_index < RC_ACTIVE; ++RC_index)
      {
       /* Compute the total sum */
       fscanf(RC_desc[RC_index], "%d", &datum);
       sum = sum + datum;
      }
 close_RC_files();

 /* Convert the integer value to a numeric string */

 itoa(sum, result);

 /* Generate the reply message */

 strcpy(reply, "VALID SUM = ");
 strcat(reply, result);
}

/*****************************************************************************/
```

```
co_RECORDS()
/* Coalesce the relation fragments from the RCs */
{
 int compare_index;
 int lf_pos;                 /* position of line feed character   */
 int next;
 int rec_cnt;                /* counts records read from each RC */
 int rec_limit;              /* random bound for each RC         */
 int RC_index;

 buf_index = Ø;

 /* SIMULATE retrieving data from each RC via RC data files */

 open_RC_files();
 for(RC_index = Ø; RC_index < RC_ACTIVE; ++ RC_index)
      {
       rec_limit = random(MAX_RC_DATA);
       for(rec_cnt = Ø; rec_cnt < rec_limit; ++rec_cnt)
          {
           fgets(RC_buffer[buf_index].data,MAX_DATA_LENGTH,RC_desc[RC_index]);
           /* Remove the linefeed character added by fgets from each record */
           lf_pos = strindex(RC_buffer[buf_index].data, "\n");
           RC_buffer[buf_index].data[lf_pos] = '\Ø';
           ++ buf_index;
          }
      }
 close_RC_files();
```

```
  /* Now coalesce the relation fragment records from all of the RC's      */
  /* Take each record and remove its duplicates in the rest of the buffer */

  buf_limit = buf_index;
  compare_index = 0;
  while(compare_index < buf_limit)
       {
        strcpy(compare_buffer, RC_buffer[compare_index].data);

        /* Go through the rest of the buffer and remove duplicates */

        buf_index = compare_index + 1;
        while(buf_index < buf_limit)
             {
              if (strcmp(compare_buffer, RC_buffer[buf_index].data) == 0)
                 {/* A duplicate relation fragment exists   */
                  /* Remove the duplicate by overwriting it */
                  next = buf_index + 1;
                  while(next < buf_limit)
                       {
                        strcpy(RC_buffer[(next-1)].data, RC_buffer[next].data);
                        ++ next;
                       }
                  -- buf_limit;
                 }
              else
                  ++ buf_index;
             }
        ++ compare_index;
       }

 /* Generate a reply message */

  if (buf_limit > 0)
     {
      strcpy(reply, CONTINUE);
      strcat(reply, "VALID REQUEST:");
     }
  else
     strcpy(reply, "VALID REQUEST -- No Data");
 }

 /***********************************************************************/
```

```
bad_req()
/* Generates an error message */
{
 strcpy(reply, "INVALID REQUEST ");
}

/***************************************************************************/

send_results()
/* This procedure transmits the results of a request to the host    */
/* over the interprocess channel established by the socket facility */
{
 if (reply[Ø] == CONTINUE[Ø])
    {/* Send coalesced relation fragments to the host from RC_buffer*/
     send(channel, reply, RES_SIZE, Ø);
     for(data_sent = Ø; data_sent < buf_limit; ++data_sent)
          {
           send(channel, RC_buffer[data_sent].data, MAX_DATA_LENGTH, Ø);
          }
     send(channel, TERMINATE, 2, Ø);
    }
 else
     send(channel, reply, RES_SIZE, Ø);
}

/*****************************************************************************/

yyerror(s)  /* Error routine used by yyparse() (see pg. 18 in YACC booklet)*/
            /* Causes the YACC parser to display an error message and must */
            /* be provided.                                                */
char *s;
{
 fprintf(stderr, "%s\n",s); /* Print error message from YACC parser */
}

/*****************************************************************************/

yywrap()   /* Used by Lex in yylex() at end of input data */
           /* Causes lex to finish and must be provided   */
{
 return(1);
}
```

APPENDIX J

C LANGUAGE SOURCE CODE FOR THE GLOBAL CONSTANTS, FUNCTIONS, AND VARIABLES

```
/*                              GLOBAL CONSTSANTS                              */

#define MAX_ATTRS       10      /* max number of attrs in a relation  */
#define MAX_AUTO        10      /* number of possible string values   */
#define MAX_GEN         10      /* total requests made automatically  */
#define MAX_DATA_LENGTH 80      /* MAX string size of RC data         */
#define MAX_RC_DATA     10      /* total data recs in RC data files   */
#define MAX_RELS        10      /* max number of relations in database*/
#define MAX_REQ         10      /* max requests in a req file         */
#define MAX_WHERE       100     /* max length of a WHERE clause       */
#define NAME_SIZE       20      /* max length of a name               */
#define QUIT            0       /* value to terminate a user's menu   */
#define RC_ACTIVE       4       /* current RCs in RRDS configuration  */
#define REQ_LENGTH      300     /* maximum length of request field    */
#define RES_SIZE        80      /* length of Controller's reply       */
#define VAL_SIZE        20      /* max length of a string variable    */
```

```
/*                          GLOBAL  VARIABLES                                    */

int  active_bus;                 /* 1 if channel "busl" is connected              */
char BLANK[2];
char BOR[2];                     /* beginning of request delimiter                */
char CONTINUE[2];                /* indicates more data follows                   */
char EOR[2];                     /* end of request delimiter                      */
char TERMINATE[2];               /* indicates end of data transmission            */
FILE *file_desc;                 /* ptr to a request file                         */
char file_name[NAME_SIZE];       /* name of request file                          */
int  new_line;                   /* dummy variable for input responses            */
FILE *out_desc;                  /* ptr to a output file                          */
char out_file[NAME_SIZE];        /* results of processed reqs                     */
char output[2];                  /* directed output(screen,file,both,S/F/B) */
FILE *RC_desc[RC_ACTIVE];        /* data file ptr to backend RC's                 */
char recv_request[REQ_LENGTH];   /* Controller's current request buffer           */
int  req_cnt;                    /* number of requests read from file             */
char reply[RES_SIZE];            /* Controller's reply to a request               */
int  make_channel, channel;      /* stores pipe info for 'busl'                   */
int  tmpl_cnt ;                  /* number of templates read from file            */
char tmpl_name[NAME_SIZE];       /* name of TEMPLATES file                        */
FILE *tmpl_desc;                 /* ptr to the TEMPLATES file                     */

FILE *fopen();                   /* declaration of fopen function                 */

struct {
        char request[REQ_LENGTH];
       }req_buffer[MAX_REQ];

/*  Structure variables for relation templates  */

struct attribute {
                  char at_name[NAME_SIZE];
                  char at_type;
                 };

struct relation  {
                  char rel_name[NAME_SIZE];
                  int  rel_nof_attrs;
                  struct attribute rel_attrs[MAX_ATTRS];
                 };

struct relation rel_templates[MAX_RELS];

/* struct variable for 'busl' channel info */

struct sockaddr_un soc_name={AF_UNIX,"busl"};

/* variable to store relational operators */

struct {
         char OP[3];
       } REL[6];
```

```
/*************************************************************/

itoa(n,s)      /* convert integer in n to characters in s */
               /* System function could not be found      */
int n;
char s[];
{
 int c, i, j, sign;

 if ((sign = n) < Ø) /* record sign */
      n = -1 * n;         /* make n positive */
 i = Ø;
 do {  /* generate digits in reverse order */
     s[i++] = n % 1Ø + 'Ø';  /* get next digit */
    } while (( n /= 1Ø) > Ø); /* delete it */
 if (sign < Ø)
    s[i++] = '-';
 s[i] = '\Ø';

/* Now reverse the string */

 for (i = Ø, j = strlen(s)-1; i < j; i++, j--)
     {
      c = s[i];
      s[i] = s[j];
      s[j] = c;
     }
}

/*****************************************************************/

strindex(s,t)  /* return first position of t in s, -1 if none */
char *s, *t;
{
 int i, j, k;
 for(i = Ø; *(s + i) != '\Ø'; i++)
    {
     for(j = i, k = Ø;
         *(s+j) != '\Ø' && *(t+k) != '\Ø' && *(s+j) == *(t+k); j++, k++)
         ;
     if(*(t + k) == '\Ø')
       return(i);
    }
 return(-1);
}

/*****************************************************************/

random(modifier)
int modifier;
{
 return(rand() % modifier);
}

/*****************************************************************/
```